UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERIA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PROYECTO DE FIN DE CARRERA

Aplicación Web para la selección y clasificación de contenido en la red social Twitter

Autora: Marta Fuentes Lara

Tutor: Pablo Alejandro Acuña Ruano

Octubre, 2011

Título: Aplicación Web para la selección y clasificación de contenido en la red social Twitter.

Autora: Marta Fuentes Lara.

Director: Pablo Alejandro Acuña Ruano.

EL TRIBUNAL

Presidente: Telmo Zarraonandia

Vocal: Derick Leony

Secretario: Mario Rafael Ruiz Vargas

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 27 de Octubre de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

En primer lugar expresar mi agradecimiento a mi tutor Pablo, por ofrecerme la oportunidad de llevar a cabo este proyecto, por su dedicación y por su apoyo.

De la misma manera, dar las gracias a mis compañeros de la Universidad Carlos III de Madrid, por sus consejos y apoyo a lo largo de toda la carrera.

Y por último, mencionar a mi familia. Gracias por confiar en mí, por vuestro apoyo y sobre todo por vuestra comprensión durante estos 5 años.

Resumen

Que hoy en día existe muchísima información disponible en Internet para los usuarios no es algo nuevo, pero de aquí a unos años atrás se ha descubierto una fuerte tendencia al uso de las redes sociales como Twitter, la cual es altamente partícipe de que la información vaya aumentando cada día. Ante la afluencia de tal cantidad de información se complica la búsqueda y clasificación de lo que es verdaderamente relevante para el usuario. De esta dificultad nace este proyecto, cuyo objetivo principal es facilitar al usuario de la red social Twitter el filtrado y clasificación de la información.

Este proyecto es una aplicación Web implementada en HTML 5 que consiste en permitir al usuario realizar búsquedas de contenido en Twitter. Este contenido viene representado en la aplicación como *'tweets'*, es decir, un conjunto de información compuesto de: un texto, un enlace al usuario de Twitter que lo creó y una imagen del mismo, entre otros campos. Además, estos *tweets* aparecerán ordenados cronológicamente del más reciente al más antiguo cuando se realice una búsqueda.

La funcionalidad que ofrece esta aplicación se centra principalmente en tres acciones:

La primera consiste en permitir arrastrar los tweets a dos zonas de clasificación que son personalizables, ya que permiten fijar un título de clasificación y una reorganización de los tweets ya clasificados.

Por otro lado, la aplicación cuenta con botones de borrado para el descarte de la información, permitiendo eliminar los tweets de una zona de clasificación, de las dos o eliminar los tweets de manera individual.

Y por último, se da la posibilidad al usuario de descargar la información clasificada, en un fichero con formato 'pdf' que contendrá la información de los *tweets* junto con el título de clasificación que haya fijado el usuario.

Todo ello contando con un sistema de almacenamiento local que se encarga de guardar la información clasificada y recuperarla para que el usuario siempre la tenga disponible.

Palabras clave: búsqueda, filtrado, clasificación y almacenamiento.

Abstract

Nowadays, it is not new that there is much information available online to users, and it recently has found a strong tendency to use the social networking site Twitter, which highly contributes in the increasing of such information every day. Taking this into account, the search and classification of information becomes very difficult in order to differentiate what is truly relevant to the user. This project is born on this difficulty, with the main objective to provide of mechanisms for filtering and classification of information to Twitter users.

This project is a web application implemented in HTML 5 which allows the user to search content on Twitter. This content is represented in the application as 'tweets', i.e. a set of information consisting of: a text, a link to the Twitter user who created it and a picture of it, among others; besides, these tweets appear in a chronological order from newest to older when performing a search.

The functionality provided by this application focuses primarily on three actions:

The First action consists of allowing dragging tweets to two zones of classification that are personalized, since they allow setting a title of the classification and a reorganization of tweets already classified.

On the other hand, the application has clear buttons for the disposal of information, allowing tweets to be removed from a zone classification or remove individual tweets by dragging them to the trash.

And finally, it gives the user the possibility of downloading categorized information in a 'pdf' format file containing the information of the tweets with the title of classification laid down by the user.

For achieving this, we use a local storage mechanism that is in charge of storing categorized information and recover it to the user when is required.

Keywords: search, filtering, classification and storage.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1. Introducción

Hay mucha y diversa información disponible, y cada día que pasa hay mucha más, no solo en las redes sociales y la Web sino que en nuestro día a día también. Pero de aquí a unos años atrás se ha descubierto una nueva tendencia por las redes sociales y dentro de ellas Twitter[1] juega un papel importante.

Cada red social tiene un objetivo y un uso principal y en estos momentos Twitter es muy consumido por personas que tienen una gran relación con la comunicación de información. Por ejemplo, es de una gran utilidad para los periodistas, quienes buscan y contrastan noticias con la información disponible en Twitter.

Sin embargo, la gran cantidad de información dificulta la obtención y gestión de datos verdaderamente útiles para su utilización. De por sí resulta difícil encontrar lo que se busca en una base de datos en la que existe demasiada diversidad de información, pero igual de difícil o más es, una vez encontrado el dominio que interesa, seleccionar lo que es realmente relevante de todo ello. Este problema es el que se estudia en este proyecto, la carencia de la red social de filtrar y procesar la información obtenida, no sólo dentro de la misma aplicación sino también poder disponer de ella en otros entornos.

En este proyecto se estudia la funcionalidad que ofrece la red social Twitter, analizando todas aquellas que tengan relación con la comunicación de información al usuario y enfatizando en el método de tratamiento de la misma, tras lo cual se descubre que no cuenta con una funcionalidad específica que permita clasificar la información obtenida de sus búsquedas, por lo que se propone una se descubren deficiencias en la clasificación de información aplicación Web complementaria que cumpla con dicha expectativa.

1.1 Definición del problema

Con el despliegue de la red social Twitter surge la necesidad de un tratamiento de la información que fluye en ella. Los servicios que ofrece se basan principalmente en el acceso a la información pero no en el filtrado y clasificación de la misma.

Es por eso que se plantea qué hacer con la información obtenida y de qué manera tratarla. A consecuencia de esta deficiencia de Twitter nace este proyecto: una aplicación Web para la clasificación y el almacenamiento de contenido de la red social Twitter.

[1] http://www.twitter.com

A lo largo de todo este documento se profundizará más sobre las características y la funcionalidad que ofrecerá esta nueva aplicación.

1.2 Objetivos

El objetivo fundamental de este proyecto es minimizar el exceso de información en la red social Twitter y permitir su clasificación, puesto que la obtención de información cuando se realiza una búsqueda puede ser enorme y dificulta la selección de información necesaria e importante. En base a este objetivo principal se proponen los siguientes sub-objetivos:

- Permitir un almacenamiento temporal de la información clasificada por el usuario en el navegador para evitar pérdida innecesaria de datos.
- Guardar la información inmediatamente después de ser clasificada y sin necesidad de cargar de nuevo la página.
- Obtener la información guardada de manera automática cada vez que se realice una nueva búsqueda o cada vez que se cargue la página, para que el usuario pueda en todo momento tener visible y a su disposición la información previamente procesada.
- Tener dos zonas personalizables en las que poder clasificar información de temas específicos.
- Descartar la información clasificada o contenida en los resultados de la búsqueda. Este sub-objetivo se divide a su vez en otros tres:
  - Eliminar la información contenida en una zona de clasificación de terminada.
  - Eliminar toda la información clasificada hasta el momento.
  - Eliminar la información de cada 'tweet' de manera individual.
- Permitir la descarga de la información clasificada en un fichero con formato '.pdf'.

1.3 Estructura del documento

Después de esta breve presentación del problema y la solución que se plantea en este proyecto, se explican los diferentes capítulos de los que se compone este documento:

- **Estado del arte:** en este capítulo se presenta un análisis de las tecnologías y aspectos relacionados con este proyecto. Se detallan las posibilidades que

ofrece cada una y que podrían ser de utilidad para el desarrollo del proyecto, así como los motivos de descarte o elección de la misma.

- **Desarrollo de la solución:** aquí se especifica el método de desarrollo por el que se ha optado para la implementación de la solución propuesta. Contiene el análisis y diseño realizado para su puesta en marcha, una especificación detallada del proceso de implementación, una especificación de la aplicación final desarrollada y por último una descripción del correcto funcionamiento de la aplicación.
- **Presupuesto:** en este capítulo se lleva a cabo un presupuesto sobre el desarrollo del proyecto. Este presupuesto es una aproximación del coste económico que supondría realizar esta aplicación en la vida real. También el hacer un presupuesto ayuda a contrastar la viabilidad del proyecto.
- **Conclusiones:** en este apartado del documento se tratan los aspectos a destacar del desarrollo del proyecto. Incluye un sub-apartado en el que se explican los problemas encontrados y la solución adoptada, y además se exponen las posibles mejoras y una conclusión personal sobre el conjunto del proyecto.

Además de estos capítulos el documento incluye un apartado con la bibliografía utilizada y las referencias consultadas y un anexo en el que se explican los acrónimos utilizados en este documento.

Capítulo 2

Estado del Arte

2. Estado del arte

En este capítulo se realiza un análisis de las tecnologías que podrían ser de utilidad para desarrollar este proyecto.

Se ha pensado que la implementación podría realizarse en HTML 5 (HyperText Markup Language), un lenguaje de marcado que puede ayudar enormemente a crear y mantener una estructura sólida en la futura aplicación. El uso de este lenguaje facilitaría el reconocimiento de cada parte de la aplicación y ayudaría al desarrollador en el mantenimiento de la misma.

Pero además de HTML 5, se realiza un estudio de las tecnologías que complementan la funcionalidad que ofrece este lenguaje y que serán necesarias para la implementación de los objetivos propuestos. A continuación se resumen brevemente estas tecnologías:

- **CSS 3 (Cascading Style Sheet)** → se utiliza para definir los estilos de las páginas Web.
- **JavaScript** → permite tener un control sobre la estructura del documento que se desea procesar, además ofrece propiedades y atributos que pueden ser de interés para cumplir con el objetivo de almacenar la información. Se detallarán esas propiedades y atributos y se relacionarán con los objetivos a cumplir.
- **API's de Twitter** → Twitter ofrece interfaces que posibilitan el acceso a la información de su base de datos. Se analizarán cada una de ellas y se justificará la elección.

Después de esta breve introducción se pasa al análisis detallado de cada tecnología comentada.

2.1 HTML5

2.1.1 Introducción

¿Qué es HTML5?

HTML 5 es un lenguaje de marcado que tiene como finalidad describir la estructura de las páginas Web. Esto se realiza a través de un método de marcado el cual se basa en la utilización de etiquetas.

Este lenguaje está definido en base al DOM (Document Object Model), es decir, la representación interna de la Web con la que trabaja un navegador.

En él, están incluidos elementos también utilizados en los estándares HTML 4 y XHTML 1 (eXtensible Hypertext Markup Language) además de nuevos elementos, de los que se hablará más adelante.

Algunas de las funciones más importantes que permite este lenguaje a los diseñadores son:

- Publicar documentos en línea con títulos, listas, fotos, etc.
- Recuperar la información en línea.
- Diseñar formularios para realizar transacciones son servicios remotos.

¿Por qué es importante HTML5?

Todos estamos de acuerdo en que una página Web que sea clara y que disponga de una estructura bien definida es una página más accesible y más fácil de comprender, tanto para los usuarios como para los diseñadores a la hora de mantener e incluso actualizar la Web.

HTML[2] ya nos permitía estructurar las páginas Web pero con HTML 5[3] se han definido nuevas etiquetas semánticas que nos dan la oportunidad de diferenciar las secciones de la estructura de la página de una manera más específica. Por ejemplo, permite diferenciar la cabecera del cuerpo y del pie de la Web, que no era posible en versiones anteriores de HTML y por tanto, posibilita la creación de una estructura mejor diferenciada y definida.

Una página Web que se base en dicha estructura hará que destaque positivamente de las que no usan este formato.

Por ello, para la implementación de este proyecto se ha elegido este lenguaje que además beneficiará la tarea de los buscadores y de cualquier aplicación que necesite leer de páginas Web.

Para ver de una manera más grafica las novedades de HTML 5 con respecto a HTML, podemos observar la siguiente figura [1]:

[2] http://www.w3.org/TR/html/
[3] http://www.w3.org/TR/html5/

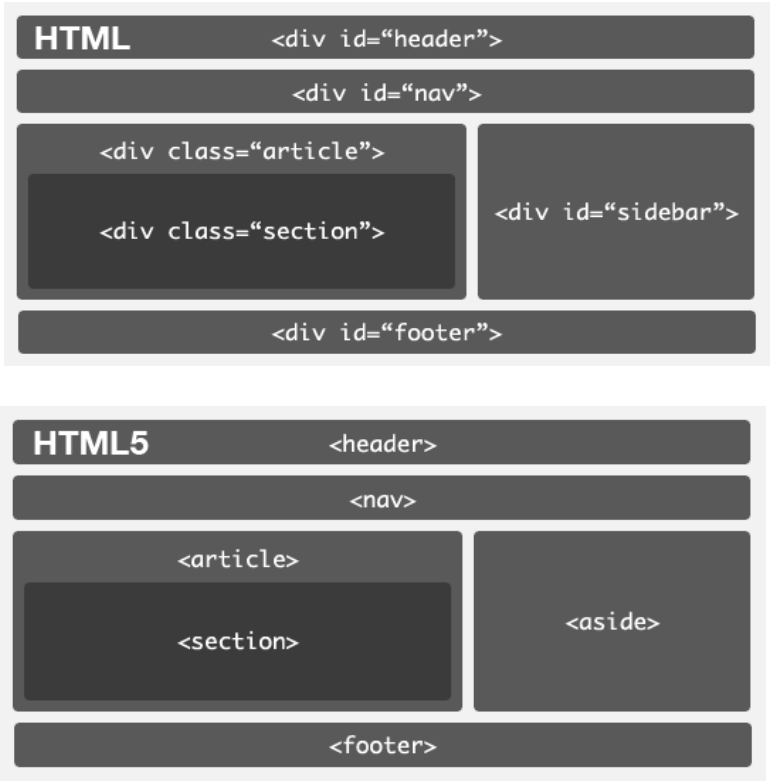


Figura 2.1. Estructura de HTML y HTML 5

2.1.2 Semántica HTML 5

HTML5 proporciona elementos que permiten describir la semántica del contenido de las páginas para permitir una mejor comprensión de su estructura. Algunos de los elementos estructurales más importantes y que diferencian esta versión del lenguaje (HTML 5) son:

- **<header>**: Sirve para delimitar la cabecera de una página Web.
- **<nav>:** Es la sección que se encarga de la navegación por el sitio Web.
- **<article>:** Con él delimitamos un contenido independiente en un documento, lo que facilitará a las herramientas la extracción de la información.
- **<section>**: Sirve para representa una sección "general" dentro de una Web. Puede contener sub-secciones y se recomienda acompañarlo de etiquetas h1-h6 para estructurar mejor toda la página. Al igual que <*article*>, facilita la extracción de la información.
- **<aside>:** es un bloque de texto en el que se puede colocar información relacionada con un texto anterior. Aunque también puede incluirse información independiente. Por ejemplo, un uso habitual de esta etiqueta con contenido no directamente relacionado es la inserción de barras laterales en una Web.

- **<footer>**: permite definir los bloques que normalmente se usan como pie de página. Contendrá información acerca de la página que no tiene nada que ver con su contenido, como el autor o empresa, modos de contacto, *copyright,* etc...

2.1.3 Sintaxis HTML 5

Existen además otros elementos y etiquetas importantes en la creación de aplicaciones Web, como son los que se describen a continuación:

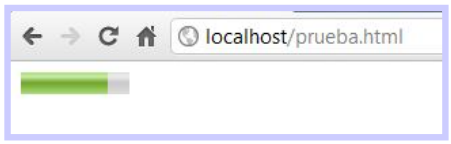
- **<dialog>**: con ella se definen conversaciones y diálogos.
- **<figure>:** se utiliza para asignar una leyenda o un título a un contenido multimedia.
- **<datagrid>:** es el encargado de representar los datos de una forma interactiva, ofreciendo la posibilidad al usuario de trabajar de una manera dinámica con ellos.
- **<command>:** se utiliza para indicar al usuario que puede ejecutar un comando en el navegador.
- **<mark>:** se utiliza para destacar un texto.
- **<meter>**: representa una medida, como por ejemplo, el número de Bytes. Depende de los atributos que se le añadan. Estos atributos pueden ser: value, min, max, low, high y optimum.
- **<time>:** sirve para las representaciones de horas o fechas.
- **<progress>**: se utiliza para representar el estado en el que nos encontramos al iniciar una tarea, por ejemplo, al iniciar la descarga de un documento.
- **<output>:** se utiliza para representar la salida de un programa.

Y por otro lado, se incorporan nuevos elementos destacables que se encargan de añadir y controlar contenido de audio y video así como permitir realizar gráficos en dos dimensiones, algunos de estos elementos son:

- **<audio> y <video>**: ayudan a delimitar de una forma sencilla donde se encuentra el contenido multimedia de la página Web.
- **<canvas>:** es el elemento que permite crear gráficos y dibujos dentro de la página Web.
- **<embed>:** se utiliza para el contenido incrustado y ejecutado por *plugins*. Tiene la ventaja de que es soportada por casi todos los navegadores.

En la siguiente figura se muestra un ejemplo del elemento <*meter*> que representa una medida dentro de un rango, el intervalo va desde 0 a 100 y en este ejemplo toma el valor 80.

```
<meter min="0" max="100" value="80">
   <span>1/3</span>
</meter>
```



Ejemplo 2.1. Representación de una medida

2.1.4 Contenido editable

Uno de los novedosos atributos que ha añadido HTML 5 es '*ContentEditable* [2]'. Este atributo da la posibilidad al usuario de editar contenido HTML 'in-situ' (en el mismo lugar).

Pero para poder editar contenido, HTML 5 también presenta otro atributo similar pero con objetivo distinto, '*designMode*'. La diferencia entre ambos está en que el primero solo permite editar aquellos elementos que tengan en la etiqueta el atributo `'<… contentEditable = "true">'` en el documento HTML así como los elementos que se incluyan dentro de este elemento, y con el segundo se puede editar todo el documento.

Dentro de estas dos alternativas en este proyecto se podría utilizar '*contentEditable*' para permitir al usuario editar solo aquella información que se obtiene de la búsqueda realizada y posibilitar la categorización y edición de la misma.

El uso de este atributo sería interesante para dar la posibilidad al usuario de destacar aquella información que se considere importante, por ejemplo señalar información en el contenido de un texto que desee localizar con facilidad.

Este atributo al igual que la gran mayoría de los nuevos elementos que añade HTML5 es soportado por los navegadores actuales en sus últimas versiones, como Firefox[4], Google Chrome[5], Internet Explorer[6], Opera[7] y Safari[8], lo que es importante para obtener una mayor aceptación por parte de aquellos usuarios que utilicen estos distintos navegadores.

El principal objetivo de este atributo ('ContentEditable') es permitir al usuario editar información contenida en la página Web y que se genere y mantenga automáticamente la actualización del documento HTML 5 con los cambios producidos por el usuario. De esta manera también se facilita al diseñador de la

[4] http://www.mozilla.com/firefox
[5] http://www.google.com/chrome?hl=es
[6] http://windows.microsoft.com/es-ES/internet-explorer/products/ie/home
[7] http://www.opera.com/
[8] http://www.apple.com/es/safari/

página Web el mantenimiento de la misma, ya que cada vez que se utiliza el atributo 'ContentEditable' y se cambia el contenido de la Web, el navegador actualiza automáticamente el código HTML por detrás, sin necesidad de que el desarrollador participe en la implementación de esa actualización. Sobre esto, es importante destacar que el código que se genera puede variar en función del navegador desde el que se esté realizando los cambios del contenido en la Web.

2.1.5 Formularios

¿Qué es un formulario?

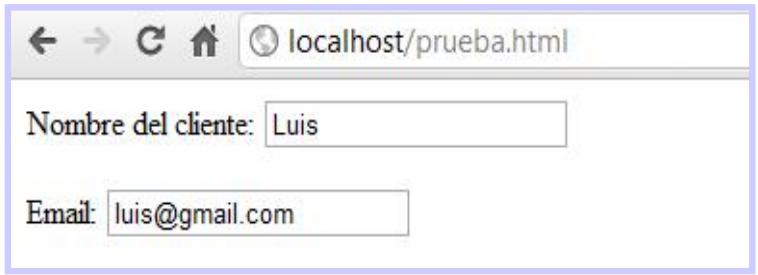
Un formulario es un componente de las páginas Web que se utiliza normalmente para almacenar información introducida por el usuario. Por ello, es imprescindible que el usuario interactúe con él, ya sea para insertar texto en un campo, seleccionar entre un conjunto de opciones, pulsar un botón o cualquier otra acción relacionada con procesamiento de datos.

La estructura básica de un formulario en un documento HTML es:

```
<form>
   <label>
      <p> Nombre del cliente: <input type="text" /> </p>
      <p> Email: <input type="email" /> </p>
   </label>
</form>
```

Donde para indicar el inicio del formulario se usa la etiqueta 'form' y dentro de la cual se incluyen los controles, creando los distintos campos del formulario representados con la etiqueta 'input' y con sus tipos correspondientes. Además opcionalmente se puede usar la etiqueta 'label' que sirve para describir un campo insertado en un formulario.

Dicha forma se representaría en un navegador de la siguiente manera:



Ejemplo 2.2. Representación básica de un formulario

Como se ha mencionado, en un formulario lo más habitual es que se utilice el elemento de introducción de datos, dicho elemento se corresponde con la etiqueta

<*input*> y ha sido ampliado satisfactoriamente en esta versión de HTML (HTML 5).

A continuación se muestran las mejoras realizadas a dicho elemento, que se encargan de concretar el tipo de dato esperado en un formulario [3]:

- En primer lugar se pueden definir entradas relacionada con el tiempo y las fechas, lo permite una mejor organización de la información, se tiene:
  - **date** → para introducir una fecha.
  - **datetime** → para poner una fecha y una hora.
  - **datetime-local** → para poner una fecha y una hora (local).
  - **month** → para especificar un mes.
  - **week** → para concretar una semana.
  - **time** → para introducir una hora.
- Por otro lado para indicar un tipo numérico, se tiene:
  - **number** → que incluye números negativos y con decimales, estos últimos separados por un punto.
- Para delimitar un rango de números se usa:
  - **range** → que tiene el mismo funcionamiento que el tipo number pero en la web aparece en formato slider.
- Se indica que se introduce una dirección de correo electrónico con:
  - **email** → que acepta el atributo 'multiple' para pode introducir varias direcciones de correo. Estas tiene que aparecer separadas por comas.
- Si lo que se quiere es indicar una URL se usa:
  - **URL** → no tiene porqué ser 'http o https'.
- Para realizar una búsqueda se cuenta con el tipo:
  - **search** → similar al tipo texto pero con apariencia distinta.
- Para seleccionar un color se tiene:
  - **color** → que da la posibilidad de seleccionar un color de una tabla de colores.
- Para terminar, se puede introducir un teléfono con el tipo:
  - **tel**

Un ejemplo sencillo de un formulario en el que se podrían utilizar algunos de estos tipos es:

```
<form>
   <label>
```

```
        <p> Nombre: <input type=“text” /></p>
        <p> Fecha de nacimiento: <input type=“date” /></p>
        <p> Edad: <input type=”number” /></p>
        <p> Teléfono: <input type=”tel” /></p>
        <p> Email: <input type=“email” /></p>
        <input type="button" method=”post” value="Enviar" />
    </label>
</form>
```

Ejemplo 2.3. Representación de un formulario con distintos tipos de entradas.

Otra forma básica en los formularios son las leyendas (<legend>), que permiten al usuario elegir entre distintas opciones, por ejemplo, el género.

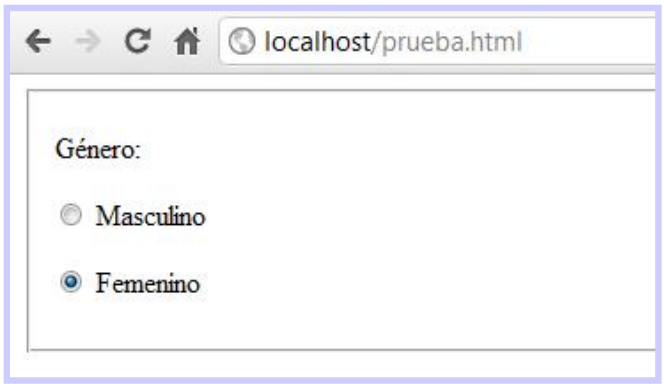
La estructura más habitual de este tipo de formas es la que se muestra a continuación:

```
<form>
  <fieldset>
    <p><legend>Género:</legend></p>
    <p><label>
      <input type=”radio” name=”genero” /> Masculino</label>
    </p>
    <p><label>
      <input type=”radio” name=”genero” /> Femenino</label>
    </p>
  </fieldset>
</form>
```

Como en el caso anterior se determinan los controles con ‘label’ y los distintos campos con ‘input’ y además se crea un tipo radio para determinar las distintas opciones de la leyenda como botones de radio, se les da un nombre común con

'name=genero' para poder tomarlos como un grupo y entonces formamos la leyenda asociada con 'legend'.

Visualmente se representaría en el navegador así:

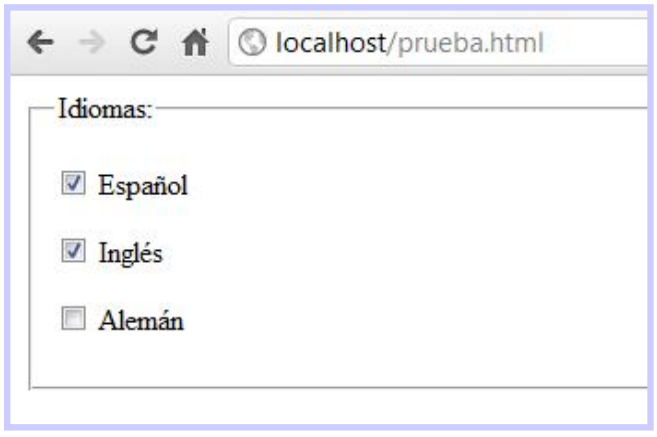


Ejemplo 2.4. Representación de una leyenda.

Del mismo modo podemos crear casillas de verificación en las que podremos elegir más de una opción a la vez, lo que se haría en HTML 5 con el tipo 'checkbox', veamos el código que lo generaría:

```
<form>
  <fieldset>
    <legend> Idiomas: </legend>
    <p><label><input type="checkbox" /> Español </label></p>
    <p><label><input type="checkbox" /> Inglés </label></p>
    <p><label><input type="checkbox" /> Alemán </label></p>
   </fieldset>
</form>
```

Quedaría:



Ejemplo 2.5. Representación de casillas de verificación.

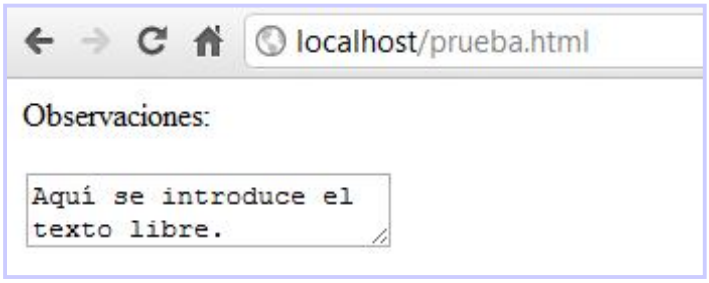
Por otro lado, HTML 5 presenta el tipo '*textarea*' para crear un campo en un formulario en el que poder introducir texto libre, un ejemplo típico de este

elemento sería el campo 'Observaciones' de un formulario para notificar una incidencia.

Para representar este ejemplo debemos utilizar el siguiente código HTML:

```
<form>
      <p> Observaciones: </p>
      <label><textarea></textarea></label>
</form>
```

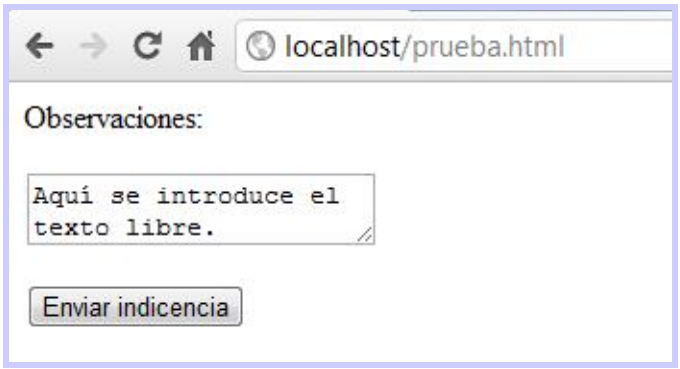
Cuyo código aparecería en la Web:



Ejemplo 2.6. Representación de cuadro de texto.

Por ultimo, tenemos a nuestro alcance botones para realizar envíos, por ejemplo para seguir con el ejemplo anterior, creamos un botón para enviar una incidencia.

```
<form id="ejemploForm" method="post" action="prueba.html">
      <p> Observaciones: </p>
      <p> <label><textarea></textarea></label></p>
      <input type="submit" value="Enviar incidencia" />
</form>
```



Ejemplo 2.7. Representación de envío de incidencias

Comunicación entre el servidor y el formulario:

Lo esencial de los formularios en proyectos Web tiene que ver con comunicar la información introducida en el formulario a un servidor para almacenarla, consultarla y/o tratarla. Esto se puede llevar a cabo con peticiones GET o POST.

Es importante mencionar que la información no es enviada tal cual se escribe en los formularios sino que se envía codificada.

Utilizando el ejemplo del apartado anterior a continuación se explica como se implementaría la comunicación de dicho formulario con un servidor:

```
<form id=”ejemploForm” method="post"
enctype="DatosAgrupadosDelFormulario" action="prueba.html">
   <p> Observaciones: </p>
   <p>
         <label><textarea name=”observaciones”></textarea></label>
   </p>
   <input type=”submit” value=”Enviar incidencia” />
</form>
```

Con el atributo ‘method = “post” ’ indicamos el método que utilizaremos para el envío de las peticiones y con ‘enctype’ especificamos que los datos enviados son codificados e indicamos la dirección del servicio que se encargará de ellos.

Consideraciones importantes para el relleno de un formulario:

A la hora de crear un formulario es necesario definir un conjunto de campos que conforman la información mínima que es imprescindible para que los datos enviados sean de utilidad para el servidor. Para que no sea posible el envío del formulario sin dicha información, en la implementación añadimos el atributo ‘*required*’ al elemento que sea obligatorio introducir.

También es interesante comentar el atributo ‘*maxlength*’ que se añade al atributo ‘textarea’ para delimitar la cantidad máxima de letras que se pueden escribir en la caja de texto libre.

Un ejemplo representativo de esto sería definir el nombre de la persona que realiza un pedido como campo obligatorio y fijar una longitud máxima de la caja de texto a 150 caracteres:

```
<form  method="post" enctype="DatosAgrupadosDelFormulario"
action="DireccionDeLaAplicacionEncargadaDeLosDatos">
   <label><p> Nombre: <input name=”nombre” required /></p></label>
   <p> Observaciones:
      <label>
      <textarea name=”observaciones” maxlength=”150”> </textarea>
      </label>
   </p>
   <button type=”button”> Enviar incidencia </button>
</form>
```

2.2 CSS3

2.2.1 Introducción

¿Qué es CSS?

CSS [4] es un lenguaje para definir el diseño de las páginas Web que están realizadas con HTML. El objetivo principal de este lenguaje es separar la apariencia del contenido de la Web para facilitar el mantenimiento y actualización de la misma, por este motivo CSS y HTML son independientes el uno del otro.

CSS 3 [5] es una nueva especificación de CSS que da la posibilidad a los diseñadores de páginas Web de hacer páginas más dinámicas y elaboradas que con versiones anteriores.

¿Por qué es importante CSS 3?

Esta nueva especificación de CSS permite tener un mayor control sobre el estilo de las páginas Web y facilita el trabajo de los diseñadores para conseguir el aspecto que desean, permitiendo ahorrar tiempo y hacer más sencillo el código fuente, consiguiendo que las páginas sean más ligeras e intuitivas.

El diseño de una página Web es lo primero que percibe el usuario y se encargará de causarle las sensaciones y sentimientos mientras navega por la misma, por este motivo es tan importante hacer un buen diseño como facilitar su realización.

2.2.2 Transformaciones en CSS 3

Una de las nuevas aportaciones de CSS 3 es el atributo '*transform*'. Con este atributo se pueden transformar los elementos de las páginas Web haciendo que cobren más dinamismo. En este proyecto pueden ser de utilidad para crear un efecto de desplazamiento en algunos elementos.

Su implementación se realiza a través de la siguiente sintaxis:

```
transform:tipo(cantidad);
```

En este código, 'tranform' es la palabra reservada en CSS 3 para indicar que un elemento será transformado. Por otro lado, 'tipo(cantidad)' indica el tipo de transformación que se desea aplicar al elemento y en la cantidad concreta, dichas transformaciones pueden ser:

- **Rotate** → que se utiliza para girar los elementos un número de grados

determinados. Su código es el siguiente:

```
Tranform:rotate (Ndeg); donde N es el número de grados.
```

- **Skew** → se puede utilizar para inclinar un elemento, permite la inclinación tanto en coordenadas X como en coordenadas Y y el valor de la inclinación se expresa en grados.

```
Tranform:skew(gradosX, gradosY);
```

- **Scale** → permite realizar una transformación de escala del elemento de manera que podamos elegir la escala en X y en Y. La cantidad se expresa en tantos por uno. Su sintaxis es:

```
Tranform:scale(escalaX, escalaY);
```

- **Translate** → con este tipo podemos desplazar el elemento tanto en X como en Y, expresando dicha cantidad en pixeles. El código que lo implementa es el siguiente:

```
Tranform:translate (desplazamientoX, desplazamientoY);
```

Solo queda decir que distintas transformaciones se pueden aplicar a la vez a un mismo elemento, de esta manera podremos obtener los efectos que resulten de combinar estas transformaciones.

2.2.3 Propiedades estéticas de CSS 3

Como se ha comentado, CSS 3 aporta nuevas propiedades que posibilitan un mayor control del diseño del estilo y apariencia de una página Web, se comentan algunas de estas propiedades de acuerdo con la información proporcionada por 'DesarrolloWeb [6]' que serán de utilidad para la realización del estilo de este proyecto:

Bordes

Los bordes pueden ser una propiedad estética a utilizar en este proyecto, por ejemplo, en la representación de los tweets en la página Web. A continuación mostramos las propiedades más comunes:

- **border-color** → se utiliza para poner color al borde de los distintos elementos de una Web y nos permite aplicar colores distintos a cada parte, donde cada parte se corresponde con el borde de arriba, abajo, de la derecha y de la izquierda. Los valores de los colores se pueden representar con números hexadecimales o con el nombre los colores en inglés, cómo puede ser 'red', 'blue', etc.

  Por otro lado, esta propiedad puede tomar desde uno hasta cuatro valores,

cada uno de los cuales hace referencia a cada lado de un borde. Veamos las distintas posibilidades:

- Si solo se pone un valor, ese color se fijará para todos los lados del borde, por ejemplo, un borde todo de color verde sería:

```
border-color: green;
```

- Si se ponen dos valores, estos corresponderán al lado de arriba y de la derecha del borde y además se fijarán del mismo color sus bordes opuestos, por ejemplo, un borde con su lado superior azul y derecho negro, fijará también de azul el borde inferior y de negro el borde izquierdo:

```
border-color: blue black;
```

- Si se ponen tres valores, se dará color al lado de arriba, de la derecha y de abajo del borde, fijando del mismo color del borde opuesto el borde que falta. Por ejemplo, el código que se muestra a continuación asigna el color rojo al borde superior, el rosa al borde de la derecha y de la izquierda y por último asigna el blanco al borde inferior:

```
border-color: red pink white;
```

- Y la última posibilidad es si se ponen los cuatro valores. En este caso tendremos uno para cada lado del borde siguiendo el mismo patrón que en los anteriores puntos, por ejemplo:

```
border-color: #ffaa77 #ff5588 #ff77aa #ff0512;
```

También se puede optar por especificar cada uno de colores de cada lado de la siguiente manera:

```
border-top-color: red;
border-bottom-color: pink;
border-right-color: blue;
border-left-color: green;
```

Una de las ventajas que ofrece esta propiedad es su compatibilidad con la gran mayoría de los navegadores actuales.

❖ **border-radius** → esta propiedad permite la definición de bordes redondeados en las esquinas de los bordes de los elementos de una página Web.

En su implementación se debe especificar las medidas del radio de la curva, es decir, '`border-radius: 3px;`'. En esta implementación, se fijaría un redondeo de 3 pixeles a cada una de las esquinas.

De la misma manera que en la propiedad anterior, se puede ir fijando el valor de redondeo de cada una de las esquinas de una vez, donde cada valor que se da se asigna a las esquinas siguiendo el sentido de las agujas del reloj.

```
border-radius: 2px 2px 3px 3px;
```

O también podemos ir especificando el valor de redondeo de cada esquina de la siguiente manera:

```
border-top-left-radius: 2px;
border-bottom-left-radius: 3px;
border-top-rigth-radius: 2px;
border-bottom-right-radius: 3px;
```

Esta propiedad también es compatible con los navegadores actuales.

- **border-image** → con esta propiedad se pueden utilizar imágenes en los bordes de un elemento, es decir, al igual que en el caso de 'border-color' podemos seleccionar imágenes y utilizarlas en varias partes del borde, dando un nuevo estilo a las "cajas".

  La definición de esta propiedad depende del navegador que se utilice para visualizarlo, para navegadores como Mozilla se usa *moz-border-image* y para navegadores que se basan en webkit[7], webkit-border-image. El inconveniente de esta propiedad es que no está implementada en Internet Explorer.

  Para implementar esto es necesario utilizar técnicas del lenguaje HTML que añadan algún contenedor para definir los estilos CSS e imitar el borde de imagen, junto con técnicas CSS. Veamos un sencillo ejemplo:

  Partimos de un código en HTML, por ejemplo:

```
<!DOCTYPE HTML>
<link href="cssImage.css" rel="stylesheet" type="text/css">
<html>
  <head>
      <title>Imágenes en los bordes con CSS3</title>
  </head>
  <body>
    <h1>Imágenes en los bordes</h1>
    <div id="CapaDeBorde">
        Prueba de borde con imagen
    </div>
  </body>
</html>
```

Para crear el estilo de un borde con imágenes creamos el siguiente CSS asociado al HTML anterior:

```
#CapaDeBorde{
  //Para navegadores Mozilla.
  -moz-border-image: url(corazonBorde.jpg) 2 2 stretch;
```

```
    //Para navegadores Webkit.
    -webkit-border-image: url(corazonBorde.jpg) 2 2 stretch;
    //Para el navegador Opera.
    -o-border-image: url(corazonBorde.jpg) 2 2 stretch;
    padding:20px;
    width: 60px;
}
```

Donde 'url' es la dirección donde se encuentra la imagen a usar, y los siguientes valores hacen referencia al alto, ancho y como se repite la imagen. Si la imagen se repite los valores que se pueden asociar son 'stretch, round y repeat', estirado, redondo y mosaico respectivamente.

El resultado en Google Chrome sería:

Ejemplo 2.8. Representación de un borde con

Para complementar la funcionalidad de esta propiedad, CSS 3 también incluye los siguientes atributos:

- ***border-image-source*** → que permite utilizar una URL de una imagen para definir la imagen del borde.
- ***Border-image-slice*** → con este atributo indicamos el espacio de la imagen que será visible como borde en todos los lados del elemento (bottom, top, right y left).
- ***Border-image-width*** → con él podemos indicar la anchura del borde.
- ***Border-image-outset*** → utilizándole podemos determinar el área en la que la imagen se extiende mas allá del área del elemento, es decir, el área que sobrepasa el borde del elemento al que se le aplica.
- ***Border-image-repeat*** → con él podemos hacer un mosaico de imágenes en el borde del elemento.

❖ **box-shadow** → con esta propiedad podemos aplicar sombras a los distintos objetos y elementos de la Web.

Para su implementación usamos la siguiente sintaxis:

```
box-shadow: 8px -12px 1px #000000;
```

Donde el primer elemento indica el desplazamiento horizontal de la sombra, el segundo el desplazamiento vertical, el tercero el difuminado y el último es el color.

Esta propiedad es compatible con todos los navegadores actuales, pero sufre variaciones en la sintaxis en el caso de Chrome y Safari, veamos dicha variación:

```
-webkit-box-shadow: 1px 1px 1px #000008;
```

Un ejemplo visual, siguiendo el ejemplo descrito en el atributo 'border-image' es:

Ejemplo 2.9. Representación de un borde con imagen y sombra.

Cuyo código HMTL y CSS son respectivamente:

```
<!DOCTYPE HTML>
<link href="cssImage2.css" rel="stylesheet" type="text/css">
<html>
<head>
     <title> Imágenes con sombra en el borde con CSS3
     </title>
</head>
<body>
     <h1>Imágenes con sombra en el borde</h1>
     <div id="SombraCapaDeBorde">
        Prueba de sombra en el borde con imagen
     </div>
</body>
</html>
```

```
 #SombraCapaDeBorde{
   //Para navegadores Mozilla.
   -moz-border-image: url(corazonBorde.jpg) 2 2 stretch;
```

```
    //Para navegadores Webkit.
    -webkit-border-image: url(corazonBorde.jpg) 2 2 stretch;
    //Para el navegador Opera.
    -o-border-image: url(corazonBorde.jpg) 2 2 stretch;
    padding:20px;
    width: 60px;
    box-shadow: 5px 5px 0 #000f2a;
    -webkit-box-shadow: 5px 5px 0 #000f2a;
}
```

La siguiente tabla resume otras propiedades típicas en el desarrollo de la estética de una página Web con CSS 3:

Fondos	*Multiple backgrounds*	Fija varias imágenes de fondo de un elemento.
	background-origin	Posicionar una imagen de fondo a un cuadro de contenido.
	background-clip	Especifica el área de la imagen de fondo.
	background-size	Especifica el tamaño de la imagen de fondo.
Color	*Opacidad*	Se encarga del nivel de opacidad que tiene un elemento.
	colores HSL	Con ella especificamos los colores con los atributos: Color, Saturación y luminosidad.
	colores HSLA	Con ella se especifican los colores con los atributos: Color, Saturación, luminosidad y opacidad.
	colores RGBA	Añade un nuevo canal de opacidad, el canal alfa, con el cual se puede especificar la opacidad de un color.
Texto	*text-shadow*	Fija una sombra a las letras de un texto.
	text-overflow	En un cuadro de texto sustituye el contenido que no entra visiblemente en el cuadro, por '...' pero no lo elimina, solo es una manera de avisar que hay mas texto del que se muestra.
	Text-wrap	Se encarga de dividir las palabras que no entran en una línea porque son demasiado largas.
Interfaz	*box-sizing*	Sirve para especificar las dimensiones de las zonas donde se insertan los elementos ("cajas").

	nav-top, nav-right, nav-bottom, nav-left	Determina el lugar de navegación cuando se pulsan las teclas de desplazamiento.
	resize	Permite redimensionar una caja.
	outline	Permite crear bordes dobles y marcos.
Otros	*Web Fonts*	Permite utilizar fuentes en las Web que no se tengan instaladas en el sistema operativo.
	Speech	Se usa para que se el contenido de las Web pueda ser leído en voz alta por el ordenador.

Tabla 2.1. Resumen propiedades CSS 3.

2.2.4 Selectores y pseudo-clases

¿Qué son los selectores?

Los selectores de CSS 3 [8] permiten seleccionar elementos de una página Web normalmente pertenecientes a la arquitectura HTML, aunque en CSS 3 se ha logrado crear selectores que dan la posibilidad de seleccionar elementos que no son accesibles desde la arquitectura de la Web, así como seleccionar elementos cuyos atributos tengan un valor determinado.

De ellos se hablará más en detalle en apartados posteriores de este capítulo.

¿Qué son las pseudos-clases?

Las pseudo-clases en CSS 3 [9] sirven para agregar efectos especiales a los selectores. Estos selectores nos permiten dar estilo a diferentes partes de un texto, cosa que con los selectores de etiquetas no es del todo posible. Por ejemplo, en el caso de un texto que se compone de distintos párrafos a los cuales queremos poner estilos distintos, no sería suficiente con el uso de estilos para etiquetas.

Hasta la creación de las pseudo-clases solo podíamos usar los selectores para asignar estilos a las etiquetas. En cambio, con las pseudo-clases se pueden especificar estilos para partes de la Web mucho mas concretas, ya que permiten seleccionar elementos que no se reconocen fácilmente a simple vista dentro de la arquitectura de HTML y que de otro modo serían de difícil acceso. Por ejemplo, la selección del primer elemento hijo de una lista.

Este tipo de selector podría ser de mucha utilidad en una aplicación Web que ofrezca una búsqueda, ya que podría facilitar el acceso a la información resultante.

Tipos de selectores:

Existen distintas variantes de selectores [10] que nos pueden ser de utilidad a la hora de construir una hoja de estilos para un proyecto de páginas Web, algunos de los tipos más comunes de selectores que incluye CSS 3 son:

- **Etiquetas HTML** → como a cada etiqueta podemos asignarle un estilo, se le puede considerar un selector.
- **Clases** → definen estilos específicos para aplicar a las etiquetas, por tanto, es un selector. La sintaxis para la definición de una clase para un selector o etiqueta HTML es:

  `h1.letraPeq {font-size: small;}` → con ello aplicamos a cualquier etiqueta h1 con clase "*letraPeq*" una letra pequeña.

  ```
  <h1 class="letraPeq">
  ```

  `.letraPeq {font-size: small;}` → de esta forma podemos aplicar la letra pequeña a todas las etiquetas que se desee añadiendo:

  ```
  <div class="letraPeq">
  <h1 class="letraPeq">
  ```

  Para terminar, añadir que solo es posible especificar para un elemento HTML una clase, es decir, un mismo elemento solo puede tener asociado una única clase.

- **Selectores contextuales (descendentes)** → se usan para poder especificar los estilos de los elementos que se encuentren dentro de otro elemento.

  Los estilos para elementos dentro de otro elemento se definen así:

  ```
  h1 em {color: green;}
  ```

  De este modo, solo cuando las palabras enfatizadas (etiqueta '*em*') se encuentren en el contexto, es decir, dentro del elemento h1, se presentarán en verde pudiendo existir otras palabras enfatizadas con distinto estilo.

- **Selectores hijos** → se utiliza para crear estilos a los hijos de los elementos. Con hijos, se refiere a elementos directamente descendientes de un elemento y no, por ejemplo, hijos de hijos del elemento. La sintaxis de este selector siguiendo el ejemplo anterior sería:

  ```
  h1 > em
  ```

- **Selectores adyacentes** → como su propio nombre indica se utiliza para dar estilos a elementos que se encuentren a lado o próximos a otros elementos (elementos 'hermanos'), es decir, aquellos elementos que tengan el mismo padre y que aparezcan seguidos dentro de la arquitectura HTML. Por ejemplo:

  Se tiene el siguiente código HTML:

```
<body>
<header>
      <h1>Primer título </h1>
      <h2>Segundo título </h2>
</header>
      <h2>Tercer título </h2>
</body>
```

  Si se quiere fijar un estilo determinado a 'Primer título' y 'Segundo título' se haría se haría de la siguiente manera:

```
h1 + h2 {color: Pink}
```

  De esta manera solo tendrían color rosa 'Primer título' y 'Segundo título' puesto que aunque 'Tercer título' también está dentro de una etiqueta <h2> con está situado después de una etiqueta <h1>.

- **Selectores por identidad** → se le asocia un nombre único al elemento (id) que se desee asignar estilo. Se implementaría de la siguiente manera:

```
<h1 id="nombreUnico">
```

  Con esto ya tenemos al elemento identificado, después se le asigna el o los estilos de la siguiente manera:

```
#nombreUnico {
   Font-size: small;
   Color: red;
}
```

  Todos estos selectores se pueden agrupar separándoles por comas, siguiendo el ejemplo de los selectores contextuales:

```
h1 em, h2 em, h3 em {color: green;}
```

  De esta manera todas las palabras con etiqueta 'em' estarán en color verde siempre que se encuentren dentro de las etiquetas h1, h2 o h3.

  Además de estos tipos de selectores incluidos en CSS 3 pero ya existentes en CSS 2.1, se han añadido estos nuevos selectores:

- **Selectores de atributos** → dentro de los selectores de este tipo que ya existían, se han incluido estos tres selectores los cuales se encargan de:

1. Seleccionar los elementos que tengan un atributo determinado y cuyo valor empieza por una cadena indicada. La sintaxis general es:

```
Elemento[nombreDelAtributo^="valorDeLaCadena"]
```

2. Seleccionar los elementos que tengan un atributo determinado y cuyo valor termina por una cadena indicada. Su sintaxis es:

```
Elemento[nombreDelAtributo$="valorDeLaCadena"]
```

3. Seleccionar los elementos que tengan un atributo determinado y cuyo valor contiene en algún lugar el texto de la cadena que se indica. La sintaxis es similar a la de los anteriores selectores:

```
Elemento[nombreDelAtributo*="valorDeLaCadena"]
```

- **Selector general de hermanos →** También se cuenta con otro selector añadido en CSS3, el selector general de hermanos, que es similar al selector adyacente que existía antes de CSS3 con la diferencia que este nuevo selector hace referencia a los elementos que están consecutivos a otros elementos pero no solo los consecutivos en el código HTML sino también, por ejemplo, los que se encuentran al lado en el diseño físico de la Web. Por ejemplo:

  Partiendo de la estructura HTML siguiente:

```
<body>
      <h1> Primer título </h1>
      <h2> Segundo título </h2>
      <div>
            <h2> Tercer título </h2>
      </div>
      <h2> Cuarto título </h2>
</body>
```

  Si se quieren seleccionar todos los elementos con etiqueta <h2> menos el correspondiente a 'Tercer título' y ponerles un color negro, utilizaríamos el selector general de elementos hermano implementado de la siguiente manera:

```
h1 ~ h2 {color: black}
```

  Con este código excluimos a 'Tercer título' porque no tiene el mismo padre y por tanto no es hermano de <h1>.

- **Pseudos-clases y pseudos-elementos →** permiten dar estilo a enlaces, o a ciertos elementos como la primera línea de un párrafo o la primera letra de una palabra, y similares partes del contenido de la Web.

  Por un lado, tenemos los pseudo-elementos que son los elementos de una Web que no pueden ser seleccionados, es decir, que no poseen identidad propia.

Pero estos elementos si que pueden ser identificados por reglas de estilo. Los pseudo-elementos son los siguientes:

- **::first-letter** → permiten asignar estilo a la primera letra de un texto. (anterior a CSS 3)
- **::first-line** → permiten asignar estilo a la primera línea de un texto. (anterior a CSS 3)
- **::before** → selecciona la primera posición del contenido del elemento. (anterior a CSS 3)
- **::after** → selecciona la última posición del contenido del elemento. (anterior a CSS 3)

Estos cuatro pseudo-elementos son anteriores a CSS 3 y aunque en CSS 3 se mantienen, su sintaxis ha sido modificada levemente, intercambiando ":" (en CSS 2.1) por "::".

Pero además de los cambios en estos pseudo-elementos, CSS 3 añade el siguiente pseudo-elemento:

- **::selection** → que hace referencia al texto que es seleccionado por el usuario, dando la posibilidad, por ejemplo, de cambiar el color del fondo de la selección. En el uso de este pseudo-elemento es aconsejable añadir los prefijos -moz- y -webkit- para evitar posibles incompatibilidades con los navegadores.

Por otro lado, las pseudo-clases son las encargadas de clasificar los elementos de acuerdo a su estado, y se puede perder o adquirir una pseudo-clase dependiendo de su posición en la estructura del documento y de la interacción del usuario con la página Web. Se han incorporado en esta versión de CSS (CSS 3) las siguientes pseudo-clases:

- **:nth-child(N)** → Con esta pseudo-clase podemos seleccionar los elementos que ocupen la posición N dentro de los elementos hijo de un elemento padre determinado. La implementación sería:

```
.nombreDeLaClasePadre:nth-child(posición){acciones a dichos
elementos hijos}
```

- **:nth-last-child(N)** → Selecciona los elementos que ocupen la posición N de una lista de elementos en los que se toma como primer elemento de la lista el elemento que ocupa la ultima posición de la misma. Su sintaxis es:

```
.nombreDeLaClasePadre:nth-last-child(posición){acciones a
dichos elementos hijos}
```

- **:nth-of-type(N)** → Seleccionan los elementos hijos de una clase padre

que tienen el tipo especificado en la sintaxis:

```
tipoDelElemento:nth-of-type (posición){acciones a dichos
elementos hijos}
```

- **:nth-last-of-type(N)** → Como en el caso anterior se seleccionan los elementos hijos de una clase padre que tienen un tipo concreto pero se toma como referencia el último elemento de la lista.
- **:last-child** → Selecciona el último elemento hijo de una lista dentro de una clase padre.

```
.nombreDeLaClasePadre:last-child {acciones a dichos
elementos hijos}
```

- **:first-of-type** → Se encarga de seleccionar el primer elemento de la lista de hijos de un elemento padre que tengan un tipo especifico. Veamos su sintaxis:

```
tipoDelElemento:first-of-type {acciones a dichos elementos
hijos}
```

- **:last-of-type** → Selecciona el ultimo elemento hijo de la lista de un elemento padre que tiene un tipo especifico. Su sintaxis en la que sigue:

```
tipoDelElemento:last-of-type {acciones a dichos elementos
hijos}
```

- **:only-child** → Selecciona el elemento solo si se cumple la condición de que es el único elemento hijo.
- **:only-of-type** → Selecciona el elemento solo si se cumple la condición de que es el único elemento hijo de un tipo especifico.
- **:root** → Se utiliza cuando deseemos seleccionar el elemento raíz de un documento.
- **:empty**→ Se encarga de seleccionar los elementos sin hijos.
- **:enabled** → Selecciona los elementos de la interfaz que estén en estado ‘enable’.
- **:disabled** → Selecciona los elementos de la interfaz que estén en estado ‘disabled’.
- **:checked** → Se encarga de seleccionar los ‘checkboxes’ o ‘radio buttons’ que se encuentren activados.
- **:not(Selector)** → Selecciona aquellos elementos que no coincidan con el selector que se especifica como atributo. Su sintaxis sería:

```
Elemento:not(.clase){acciones a realizar a los elementos
seleccionados}
```

2.3 JAVASCRIPT

JavaScript es un lenguaje de programación interpretado y orientado a objetos, que permite crear acciones en las páginas Web y que se implementa como parte del navegador.

Sus funcionalidades principales hacen posible al usuario realizar modificaciones y/o mejoras en las interfaces de usuario y proveer la interacción con las páginas Web dinámicas. Precisamente por la comentada funcionalidad, JavaScript es imprescindible para la realización de este proyecto.

A continuación se describen algunos de los elementos de JavaScript que pueden ser útiles para llevar a cabo las distintas acciones a realizar en una aplicación Web.

Algunos de los aspectos más importantes que se verán en los siguientes sub-apartados tienen que ver con el acceso al contenido de documentos HTML (HTML DOM), los distintos formatos para el intercambio de información entre aplicaciones (XML, JSON y Native JSON), los diferentes mecanismos de almacenamiento local de la información (LocalStorage, Web SQL y IndexedDB), y por último se describe el soporte que ofrece la especificación 'Drag and Drop' de JavaScript en la estructura de una Web.

2.3.1 HTML DOM

¿Qué es DOM?

DOM (Document Object Model) es una API cuyas funciones permiten de una manera más sencilla la manipulación del contenido de los documentos XML de forma rápida y eficiente, puesto que son especialmente difíciles de procesar. También permite la misma función para documentos HTML y XHTML.

DOM esta disponible en la mayoría de los lenguajes de programación más usados y se encarga de convertir el contenido de un documento, en una estructura de nodos en forma de árbol que facilite el reconocimiento de los elementos del documento y permita un acceso controlado a los mismos.

Para poder trabajar adecuadamente con el contenido de los documentos que va a procesar DOM y poder crear dicha estructura, el contenido de la Web debe estar cargado completamente, tras lo cual se puede proceder a la transformación del contenido en un árbol de nodos estructurado que representa dicho contenido y sus relaciones [11].

¿Por qué es importante DOM?

Esta API es importante para todo programador Web ya que ofrece un control bastante preciso sobre la estructura del documento que se desea procesar, permitiendo la creación, modificación, eliminación y reemplazo de cualquier nodo de la estructura.

Además su creación fue muy importante de cara al diseño e implementación Web, puesto que permitió homogeneizar la implementación del HTML dinámico en los diversos navegadores Web, posibilitando la modificación del contenido Web sin tener que recargar la página entera.

Funcionalidades

Es importante volver a destacar que la funcionalidad principal de DOM es definir las estructuras de un documento en un lenguaje determinado.

Una vez que se tiene la estructura de un documento es necesario poder acceder a los diferentes nodos de la misma. Este acceso se realiza con HTML DOM, el estándar que permite acceder a los objetos y propiedades contenidos en la estructura de un documento HTML a través de JavaScript.

La funcionalidad básica de HTML DOM [12] consiste en obtener el objeto que se encuentra en el nodo raíz de la estructura de la página Web. Esto se implementa de la siguiente manera:

```
var objeto = document.documentElement;
```

Por otro lado, otra funcionalidad básica es obtener los nodos hijos de un nodo, lo que se puede realizar con las funciones firstChild y lastChild de esta manera:

```
var objeto = ObjetoPadre.lastChild;
```

Pero para poder acceder a cualquier nodo de manera general se utiliza el método childNodes:

```
var objeto = objetoPadre.childNodes[n];
```

Donde “n” es la posición que ocupa en la lista de los nodos hijos. Si no conocemos el número de nodos hijos que tiene un nodo podemos calcularlo de la siguiente manera:

```
var numeroHijos = objetoPadre.childNodes.length;
```

HTML DOM también ofrece la posibilidad de conocer el tipo de un nodo concreto tan fácilmente como mostrar el atributo ‘nodeType’.

Con respecto a los atributos de los elementos, el DOM de HTML (HTML DOM) permite el acceso directo a los mismos, a través de la propiedad 'attributes' que es de tipo *NamedNodeMap* y a la que se puede acceder como a una matriz.

Los métodos para operar con estos atributos son:

- ***getNamedItem(nombre)*** → obtiene el nodo cuya propiedad contenga el valor 'nombre'.
- ***removeNamedItem(nombre)*** → elimina el nodo cuya propiedad contenga el valor 'nombre'.
- ***setNamedItem(nodo)*** → añade el nodo a la lista attributes, indexándolo según su propiedad.
- ***item(posición)*** → devuelve el nodo que se encuentra en la posición indicada por el valor 'posición'.

La utilización de estos métodos hace más fácil el procesamiento y modificación de los elementos HTML. Pero el uso de estos métodos no es muy eficiente puesto que en una página puede haber miles de nodos y el acceso a ellos y a sus atributos se realiza de forma jerárquica, lo que implica recorrer demasiados nodos. Por este motivo se crean las funciones de acceso directo a los nodos, algunas de ellas son:

- **getElementsByTagName("etiqueta")** → obtiene los objetos de la Web cuya etiqueta coincida con el parámetro. Esta función se puede usar de manera recursiva a los elementos que devuelve la función.
- **getElementsByName ("cadena")** → obtiene los objetos de la Web que tengan como nombre el valor de la cadena.
- **getElementById("clave")** → obtiene el objeto cuyo atributo 'id' sea igual a 'clave'.

Además HTML DOM ofrece métodos que permiten la creación, modificación y eliminación de los elementos de la página Web, es decir, de los nodos de la estructura en árbol. Estos métodos son:

- **createAttribute(nombre)** → para crear un atributo con el nombre = 'nombre'
- **getAttribute (nombreAtributo)** → para recuperar el valor de un atributo de un objeto determinado. En su sintaxis 'nombreAtributo' se refiere al nombre del atributo que se desea obtener.
- **createCDataSection(texto)** → para crear secciones, las cuales tienen un hijo de tipo texto y que contiene el valor 'texto'.

- **createComment(texto)** → para crear un nodo de tipo comentario con valor 'texto'.
- **createDocumentFragment()** → para crear nodos de tipo DocumentFragment.
- **createElement('etiqueta')** → para crear objetos con el tipo que se indica en 'etiqueta'
- **createEntityReference(nombre)** → para crear nodos de tipo EntityReference.
- **createProcessingInstruction(objetivo, datos)** → para crear nodos de tipo ProcessingInstruction.
- **createTextNode(texto)** → para crear un nodo de tipo texto con el valor de 'texto'.
- **appendChild(nodoContenido)** → para asociar a un elemento su contenido 'nodoContenido' y por tanto crearle.
- **removeChild(nodo)** → para eliminar el nodo 'nodo', y se debe invocar en el nodo padre.
- **parentNode** → para acceder al nodo padre de un nodo directamente.
- **replaceChild** → para reemplazar un nodo por otro y se debe invocar en el nodo padre.
- **insertBefore (nodoNuevo, nodoAnterior)** → para insertar un nodo delante de otro nodo concreto.

Los métodos a utilizar para permitir las funcionalidades relacionadas con tablas son los siguientes:

- **rows** → para obtener las filas de la tabla. El resultado se guarda en un arreglo (array).
- **tBodies** → para obtener los tBody de la tabla. El resultado también se almacena en un arreglo.
- **insertRow (posición)** → para insertar una fila en la posición 'posición' dentro del arreglo de filas de la tabla.
- **deleteRow** → para eliminar una fila que ocupe la posición 'posición' en el arreglo de filas de la tabla.
- **Cells** → para obtener las columnas de una fila seleccionada. El resultado se almacena en un arreglo.
- **insertCell (posición)** → para insertar una nueva columna en la posición 'posición' dentro del arreglo de columnas de la tabla.
- **deleteCell (posición)** → para eliminar una columna que se encuentra en la posición 'posición' del arreglo de columnas.

Para crear las tablas y los atributos no existe una forma especial sino que se hace de la misma manera que se crea un elemento HTML, eso sí, con el tipo 'table'.

Todos estos métodos se llaman en los elementos a los que se desee aplicar.

Todas estas funciones y propiedades comentadas son de vital importancia en la gran mayoría de los proyectos Web puesto que ofrecen la posibilidad de obtener y modificar los elementos de una página Web dinámicamente, haciendo posible la interactividad en las páginas sin necesitar una comunicación entre el navegador y el servidor.

2.3.2 XML, JSON y Native JSON

¿Qué es XML?

XML[9] (Extensive Markup Language, en inglés) [13] es un lenguaje cuyo objetivo principal es permitir compartir la información a todos los niveles y además lo hace de una manera fácil y segura. Permite la comunicación y transferencia de información entre diferentes tipos de aplicaciones y soportes, y además permite la validación de los datos y del recorrido de estructuras por lo que facilita mucho el tratamiento de los datos dando más facilidades al desarrollador.

Esta tecnología no se usa de manera individual sino que está asociada a diversas tecnologías que la complementan, como por ejemplo, HTML.

Algunos de los principales objetivos que la hacen tan importante para poder dar un paso más hacia la estandarización son:

- Enviar, recibir y procesar la información de la misma manera que HTML para poder beneficiarse de toda la funcionalidad que ofrece dicha tecnología.
- Ser extensible.
- Ser de fácil interpretación y manejo.

Por otro lado, es importante conocer el formato en que se envían los datos:

```
<cliente>
      <nombre> Javier </nombre>
      <Apellidos>
          <Apellido> García </Apellido>
          <Apellido> Moreno </Apellido>
      </Apellidos>
      <DNI> 49321557-C </DNI>
```

[9] http://www.w3.org/XML/

```
</cliente>
```

Para recuperar los datos llamamos al método 'requestXML' y a partir de ahí se accede a través de la estructura del DOM.

Por último comentar la ventaja que puede aportar esta tecnología a las aplicaciones Web, y es que en un principio los programadores tenían que procesar los datos en función del navegador que usaran, pero con XML se cuenta con un lenguaje que se encarga del manejo de los datos independientemente del navegador que se use.

¿Qué es JSON?

JSON[10] (JavaScript Object Notation) [14] se encarga al igual que XML del envió y recepción de datos en un formato concreto y es, por tanto, una alternativa a XML. Su formato es más simple que XML, ya que se asemeja bastante al formato de definición de objetos y arreglos que es utilizado en JavaScript aunque si la estructura es muy compleja se aconseja usar XML. A continuación se muestra el ejemplo anterior en este formato:

```
{'nombre': 'Javier',
'Apellidos': ['García', 'Moreno'],
'DNI': '49321557-C'
}
```

Cómo observación decir que no se utilizan variables sino que se incluyen los valores directamente

Con este método recuperamos los datos creando una variable JavaScript que represente el objeto mediante la función '*eval*' y después ya podemos acceder a la información del objeto.

Una vez analizados los dos principales formatos para el intercambio de información entre aplicaciones, se ha decidido utilizar JSON en el desarrollo de este proyecto. La elección está basada en la sencillez y facilidad de uso de JSON, pero además también mas característica a su favor, como la velocidad de carga y su buena integración con Ajax[11] (Asynchronous JavaScript and XML.).

[10] http://www.json.org/
[11] http://www.w3schools.com/ajax

¿Qué es Native JSON?

Native JSON[12] (JavaScript Object Notation) es una herramienta que tienen incluida la mayoría de los navegadores para poder analizar y evaluar valores con formato JSON. Utiliza JSON para la transferencia de datos y permite la gestión de su contenido, es decir, se utiliza para acceder y tratar valores JSON.

Se trata además de un subconjunto de la notación literal de objetos de JS y la principal característica que le ha llevado al éxito es la simplicidad.

Native JSON [15] es una de las nuevas características que presenta JavaScript y que ofrece principalmente dos beneficios añadidos a los que tenia el uso de JSON. Estos beneficios son:

- El análisis que se realiza con Native JSON es más seguro que el que se realizaba con JSON ya que no cuenta con librerías externas, permitiendo evitar ataques de ejecución al código. Por eso es mas apropiado usa JSON cuando la fuente es completamente fiable.

  En cambio con Native JSON podemos realizar las trasferencias de información con más seguridad. De hecho, no es incompatible con los objetos que tienen métodos y al intentar convertir un objeto de este tipo a JSON se produce una excepción TypeError.

- El otro beneficio es que el análisis Native JSON es mucho más rápido, y tiene la ventaja de que está disponible en todos los navegadores actuales.

Como se ha comentado con anterioridad, Native JSON es un subconjunto de la notación literal de objetos de JavaScript y con el se pueden crear objetos, estos objetos pueden tener miembros y que estos miembros contengan objetos, etc. Un ejemplo de cómo se implementaría un objeto con un miembro que contiene una matriz con dos objetos sería:

```
var objetoJSON = {"animales": [
        {"Tipo": "Vertebrados", "nombre": "caballo"},
        {"Tipo": "Invertebrados", "nombre": "caracol"},
        ]
};
```

Después se puede recuperar la información de los objetos de la siguiente manera:

```
objetoJSON.animales[0].nombre //Lo que nos devolvería “caballo”
```

Por otro lado Native JSON también nos da la posibilidad de convertir un texto en un objeto utilizando la función JSON.parse que invoca al compilador para que analice el texto y lo convierta en una estructura de objetos. Siguiendo el ejemplo

[12] https://developer.mozilla.org/En/Using_native_JSON

anterior esto se realizaría:

```
var Objecto = JSON.parse (Texto);
```

Con Native JSON también podemos realizar el proceso anterior a la inversa, es decir, convertir una estructura de objetos en un texto JSON. Este proceso se lleva a cabo a través de la función 'stringify' de la que hay que decir que no acepta estructuras de datos cíclicas. Su implementación se realizaría:

```
var Texto = JSON.stringify (Objeto);
```

Aunque también ofrece capacidades opcionales adicionales, con ellas la sintaxis quedaría:

```
Texto = JSON.stringify (objeto [, Sustituto][, space])
```

Donde 'Sustituto' puede ser una función o una matriz que se encarga de transformar o seleccionar los resultados. Cuando 'Sustituto' sea una función se pasa la clave y el valor de cada miembro y lo que se convierte en texto es el valor de retorno no el valor original. Cuando dicho argumento sea una matriz, solo los miembros con los valores clave de la matriz serán lo que se serialicen siguiendo el orden de las claves en la matriz.

Por otro lado, si no se especifican estos parámetros se añaden a la cadena todos los atributos y propiedades del objeto.

Con respecto al argumento 'space' decir que es de carácter estético puesto que lo que hace es insertar un espacio en blanco en la cadena de salida JSON con el objetivo de facilitar su lectura y comprensión.

Si dicho argumento se trata de un número, éste indica el número de espacios que se insertarán y si es un número menor que 1 quiere decir que no se deben utilizar los espacios en blanco. Este parámetro también puede ser una cadena, si es así lo que se hace es coger la cadena o los 10 primeros caracteres de la misma si los supera y usar dicha cadena como separación.

¿Por qué es importante?

En este proyecto podría ser de utilidad para procesar la información obtenida a través de las consultas y seleccionar la información realmente relevante para cumplir con los objetivos del proyecto. Esta información seleccionada de toda la que se obtiene con la búsqueda en Twitter es la imagen del usuario que publica el tweet, el link que lleva al perfil del mismo en Twitter.com, y por último el texto que contiene el tweet.

Dicha información es la que mas tarde el usuario podrá clasificar y almacenar según sus deseos.

2.3.3 LocalStorage

¿Qué es LocalStorage?

Es una base de datos de cliente basada en pares clave-valor cuya capacidad máxima de almacenaje es de 5 Mega Bytes y que además reside en el navegador del usuario y no en el servidor.

La información se guarda dentro del navegador del equipo por lo que limita el uso de los datos exclusivamente en el ordenador y navegador, aunque permite cerrar el navegador sin perder ningún dato.

Funcionamiento

Para mostrar como se realizarían las funciones básicas de una base de datos usando localstorage[13] de JavaScript [16] se tiene el siguiente ejemplo, en el que primero se comprueba si el navegador soporta 'localstorage' y si es así, añade un registro a la base de datos, después lo recupera, lo muestra y lo elimina:

```
if (typeof (localStorage) == 'undefined') {
      alert ('Su navegador no soporta localStorage.');
}
Else {
      try {
            localStorage.setItem ("Luis", "Es muy alto.");
      }
      Catch (e) {
            if (e == QUOTA_EXCEEDED_ERR) {
            alert ('ha superado la cuota!');
            }
      }
      //devuelve: "Es muy alto".
      document.write(localStorage.getItem("Luis"));
      localStorage.removeItem ("Luis"); //Elimina el registro
}
```

Como puede comprobarse el funcionamiento de esta base de datos es bastante sencillo e intuitivo. Y profundizando un poco más se puede ofrecer un almacenamiento de verdadera utilidad.

Para crear un registro que contenga una matriz con tres valores. En su implementación lo primero que se hace es definir una matriz y los valores que interesa introducir:

```
persona var = new Array ();
```

[13] http://dev.w3.org/html5/webstorage/#the-localstorage-attribute

```
var nombre = "nombre";
var ap = "apellidos";
var dni = "12345678A";
```

Lo siguiente a realizar es incluir los valores en la matriz y está en la base de datos, lo que se realiza de la siguiente manera:

```
persona.push (nombre)
persona.push (ap)
persona.push (dni)
try {
      localstorage.setItem (persona[2], persona.join (“,”));
}
Catch (e) {
      If (e == QUOTA_EXCEEDED_ERR) {
            Alert (“No se ha podido almacenar porque se ha
excedido la cuota”)
      }
}
```

Ahora obtenemos el elemento que hemos introducido puesto que por norma general es lo que se pretende con todas las bases de datos, consultar los datos y hacer algo con ellos:

```
Var dni = “12345678A”;
Var persona = localStorage.getItem (dni);
persona = persona.split (",");
var nombre = persona [0];
var Ap = persona [1];
var dni = persona [2];

document.write ('Nombre de la persona:' + nombre + '<br/>');
document.write ('Apellido de la persona:' + Ap + '<br/>');
document.write ('DNI de la persona:' + dni + '<br/>');
```

En este código cabe destacar que la función ‘getItem’ es la que se encarga de obtener los datos asociados a la clave, lo demás son traspasos de variables y visualización de las mismas.

Todas estas funcionalidades son de utilidad para hacer que la aplicación cumpla con las expectativas y permita al usuario almacenar la información de manera local y poder recuperarla y utilizarla cómo y cuándo desee.

2.3.4 Web SQL

¿Que es Web SQL?

Web SQL[14] es una especificación de HTML5 cuyo objetivo es guardar y que persista la información en una base de datos relacional. Además tiene la buena cualidad de que está embebido en el navegador Web.

Funcionamiento

Para usar esta característica lo primero que se hace es asegurarse de que el navegador la soporta y una vez comprobado se podrá usar todas las funcionalidades que aporta esta nueva característica de HTML 5. Para ello, lo primero es realizar la conexión de la aplicación Web con la base de datos, esto se realiza de la siguiente manera [17]:

```
if (window.openDatabase){
      //El navegador soporta Web SQL.
      var db = openDatabase('nombreBD', '1', 'Descripcion', 2 *
1024); //Crea y abre la base de datos.
}
Else{
      alert('El navegador no soporta Web SQL.');
}
```

Una vez que se ha abierto la base de datos es posible realizar cualquiera de las funciones asociadas a las bases de datos, como crear una tabla, realizar una consulta, introducir, borrar o modificar información, etc. eso sí debe realizarse en un lenguaje compatible con SQLite[15].

Este concepto podría ser muy importante en este proyecto ya que es esencial tener una base de datos donde el usuario pueda almacenar la información que seleccione de las búsquedas y que le permita categorizarla y realizar con ella lo que desee en cada momento.

Esta especificación se basa en tres métodos que se explican a continuación:

- **OpenDatabase** → este método es el que se utiliza para crear y abrir la base de datos. Es de fácil manejo puesto que cuando se intenta abrir una base de datos inexistente este método la crea directamente y también se hace cargo del cierre de la base de datos.

  El código que se necesita para abrir y crear una base de datos, aunque ya se ha comentado, es el que se detalla a continuación:

[14] http://www.w3.org/TR/webdatabase/
[15] http://www.sqlite.org/docs.html

```
var db = openDatabase('nombreBD', '1', 'Descripcion', 2 * 1024);
```

Donde se pasan cuatro parámetros, 'nombreDB' es el nombre de la base de datos, el siguiente parámetro es la versión de la misma, 'Descripción' es una descripción del contenido de la base de datos y por último el tamaño que se desea que tenga en Bytes.

Por otro lado 'db' es el valor de retorno que contiene los métodos de transacción y que por tanto se tendrá que usar para poder realizar las consultas SQL.

- **Transaction** → como en toda base de datos una vez que se tiene creada la estructura es necesario crear las transacciones, esto es lo que se lleva a cabo con el método 'transaction' que además controla que si una transacción falla, no afecte a la integridad de la base de datos y lo hace ignorándola y haciendo como si no se hubiera producido.

  Este método es una función que contiene la siguiente implementación:

  ```
  var db = openDatabase('nombreBD', '1', 'Descripcion', 2 * 1024);
  db.transaction(function (X));
  ```

  Esta transacción lo que hace es crear el objeto de transacción X.

  Por otro lado, existe la transacción *db.readTransaction* que solo permite la lectura de las declaraciones, en lugar de la lectura y escritura que permite db.transaction.

  Con esto ya es posible ejecutar SQL.

- **executeSql** → es el método que sirve para leer y escribir las declaraciones y ofrece unas respuestas de llamadas que procesan los resultados de las búsquedas. Esto se aplica junto con los pasos anteriores y ejecuta el objeto de transacción definido. Un ejemplo que muestre como se implementa esto es:

  ```
  var db = openDatabase('nombreBD', '1', 'Descripcion', 2 * 1024);
  db.transaction(function (X) {
  X.executeSql('CREATE TABLE tabla1 (id unique, desc)');
  });
  ```

  Con ello lo que se ha hecho ha sido crear una tabla en la base de datos 'nombreBD' con nombre 'tabla1', y en el ejemplo siguiente se inserta un registro a dicha tabla:

  ```
  var db = openDatabase('nombreBD', '1', 'Descripcion', 2 * 1024);
  db.transaction(function (X) {
  X.executeSql('CREATE TABLE tabla1 (id unique, desc)');
  X.executeSql ('INSERT INTO tabla1 (id, desc) VALUES (0, "primer
  registro insertado")');
  });
  ```

Solo queda por conocer cómo obtenemos los datos de algún usuario o fuente externa sin perder la seguridad de la base de datos, para ello cuando insertamos un registro añadimos un segundo argumento al método 'executeSql', que asigna unos datos de campo a la búsqueda de la información. Esto se implementa de la siguiente manera:

```
X.executeSql('INSERT INTO tabla1 (id, desc) VALUES (x, x)', [id,
userValue]);
```

La gran desventaja que tiene en estos momentos este sistema de almacenamiento es que desde noviembre de 2010 se ha dejado de trabajar en su especificación.

2.3.5 IndexedDB

¿Que es indexedDB?

Es otro sistema Web que sirve para permitir a los usuarios el almacenamiento de información usando pares de clave-valor, donde estos valores pueden ser objetos complejos estructurados.

Este sistema ofrece a las aplicaciones Web almacenamiento estructurado de los objetos a través de índices, a los que no teníamos acceso con 'localStorage' y 'Web SQL' en el que se realizaba el almacenamiento a través de tablas.

En él, todo se hace en el contexto de una transacción, es decir, los objetos representan índices, tablas, etc., pero todos ellos están asociados a una transacción determinada, para no poner en peligro la integridad de la base de datos.

Utiliza peticiones, que no son otra cosa que objetos que reciben los eventos de éxito o error de la transacción y también se sirve de propiedades como 'readyState' o 'result' para indicar el estado de la solicitud. Además IndexedDB[16] también utiliza eventos DOM para indicar la disponibilidad del resultado.

¿Por qué es importante?

El API de IndexedDB es importante porque su diseño esta creado para hacer mínimo el manejo de errores, lo que siempre es beneficioso y cómodo para el desarrollador y el cliente no solo en este proyecto sino en cualquier otro que se relacione con el almacenamiento Web.

Por otro lado, IndexedDB tiene como objetivo permitir el almacenamiento de grandes cantidades de información sin conexión, otro beneficio a la hora de crear

[16] http://www.w3.org/TR/IndexedDB/

aplicaciones Web.

Funcionamiento

El funcionamiento de IndexedDB [18] se basa en un patrón que es muy simple:

1. Iniciamos una transacción.
2. Realizamos una petición para realizar operaciones en la base de datos (inserción, borrado, modificación, etc.).
3. Esperar a que el evento DOM avise de que la operación se ha completado.
4. Y por ultimo, realizar lo que se desee con los resultados obtenidos.

Para crear y abrir una conexión de la base de datos con este sistema y empezar a añadir tablas lo primero es comprobar que la versión es correcta, de lo contrario se tendría que realizar un cambio de versión. A continuación se muestra como se realiza lo ya comentado:

Creamos la petición:

```
var peticion = mozIndexedDB.open ( "nombreBaseDatos" );
```

Cabe mencionar que se añade un prefijo 'moz' para el soporte de los navegadores.

Con el código anterior se crear una base de datos en el caso de que no exista y en el caso de que exista, simplemente se abre.

Después de esto el siguiente paso a realizar es añadir un control de los errores, para lo que se utiliza 'request.onerror' y 'request.onsuccess'. El esquema general para realizar estos controles es:

```
peticion.onerror = function (evento) {
   // Tratamiento del error
};
peticion.onsuccess = function (evento) {
   // Tratamiento de los resultados
};
```

Antes de continuar, es necesario guardar los resultados para realizar las operaciones pertinentes con ellos, siempre que no se haya producido ningún error. Para guardar los resultados, simplemente los almacenamos en una variable y para cumplir con la condición de que no pueden haberse producido errores se incluye en 'peticion.onsuccess' de la siguiente manera:

```
var peticion = mozIndexedDB.open ("nombreBaseDatos");
```

```
peticion.onerror = function (evento) {
   // Tratamiento del error
};
peticion.onsuccess = function (evento) {
   // Tratamiento de los resultados
   var db = peticion.result;
};
```

El segundo paso del modelo básico de IndexedDB es crear una petición que realice una operación determinada en la base de datos. A continuación se explica cómo realizar dicho paso cuando se quieren introducir nuevos datos en la base de datos:

```
var transaccion = db.transaction (["personas"],
IDBTransaction.READ_WRITE);
```

Donde la función *transaction* tiene como argumentos: una lista de objetos a la que se le aplicara la operación, una lista que si es vacía hará que se aplique la operación a todos los datos de la base de datos; el siguiente argumento tiene que ver con el tipo de transacción, pudiendo ser de solo lectura o de lectura y escritura. Si queremos que la transacción sea solo de lectura para poder ejecutar varias transacciones del mismo tipo simultáneamente con no ponerlo bastaría, en cambio si se desea una transacción de lectura y escritura se añade IDBTransaction.READ_WRITE.

Estas transacciones reciben eventos que sirven para indicar su estado, dichos eventos pueden ser: *error, abort y complete*.

El primero se produce cuando se genera un error; el segundo cuando no se ha tratado algún error, lo que hace que la transacción aborte; y el último se produce cuando todas las transacciones de la solicitud se completan.

El código para insertar datos en la base de datos se corresponde con el siguiente:

```
transaccion.oncomplete = function(event) {
      alert ("¡Datos insertados!");
};
transaccion.onerror = function (event) {
      //aquí se tratan los errores
};
var ObjectStore = transaccion.objectStore ( "personas" );
for ( var i in DatosPersonas) {
      var peticion = objectStore.add (DatosPersonas[i]);
      petición.onsuccess = function (event) {
            //por ejemplo se mostrarían los datos
```

```
        };
}
```

También podemos eliminar información de la base de datos, que se realiza de la siguiente manera:

```
var peticion = db.transaction(["personas"],
IDBTransaction.READ_WRITE)
                    .ObjectStore ("personas")
                    .delete("48655214-Y");
request.onsuccess = function (event) {
};
```

Y para terminar, otra operación que es importante, es la obtención de datos de la base de datos, aunque este proceso puede llevarse a cabo de dos maneras. La primera de estas maneras es usando la función get ():

```
var transaccion = db.transaction (["personas"]);
var ObjectStore = transaction.objectStore ("personas");
var peticion = objectStore.get ("48655214-Y");
peticion.onerror = función (evento) {
  // Tratamiento de errores.
};
peticion.onsuccess = function (event) {
      //Tratamiento de los datos, por ejemplo visualizarlos.
      alert ("El DNI es:" + peticion.result.name);
};
```

Asumiendo que los errores se controlan a nivel de base de datos nos ahorraríamos código y la implementación anterior quedaría:

```
db.transaction("personas").ObjectStore("personas").get("48655214
-Y ").onSuccess = function (event) {
      alert ( "El DNI es:" + peticion.result.name);
};
```

IndexedDB ofrece otra funcionalidad y consiste en el uso de cursores. Un cursor se utiliza cuando se desea pasar por todos los valores de los objetos de un almacén. Esto se implementaría con la función 'openCursor()' la cual tiene como objetivo general recuperar todos los objetos de un almacén e introducirlos en un arreglo. La forma común de implementar esto es:

```
var personas = [];
objectStore.openCursor().onSuccess = function (event) {
      var cursor = event.target.result;
      if (cursor) {
            personas.push (cursor.value);
```

```
            cursor.continue ();
        }
        else {
            alert ( "conseguir a todas las personas:" +
        personas);
        }
};
```

Con Mozilla Firefox también podemos realizar lo mismo con la función 'getAll()' aunque no es una regla de indexedDB.

```
objectStore.getAll ().onSuccess = function (event) {
            alert ( "conseguir a todas las personas:" +
        personas);
};
```

De estos dos métodos, es aconsejable usar 'getAll()' cuando se desee obtener una matriz de todos los objetos en un almacén de objetos pero es mucho más eficiente usar cursores cuando solo interese ver todas las claves.

Pero lo que de verdad es importante de este sistema es el uso de índices que otorga una gran eficiencia en las operaciones de búsqueda. Estos índices se implementan de la siguiente manera:

```
var indice = objectStore.index ("nombre");
indice.GET ("Juan").onSuccess = function (event) {
alert ( "El DNI de Juan es:" + event.target.result.dni);
};
```

En este caso el nombre es una clave que puede estar repetida por lo que la búsqueda devuelve el registro con la clave más baja. Pero si se necesitan todos los registros que cumplan con la condición de 'nombre = Juan' entonces es apropiado usar cursores e índices.

Conclusiones

En este proyecto se elegirá LocalStorage como opción de almacenamiento de la información debido a la facilidad de uso, a su potencia y a que tiene soporte en la mayoría de los navegadores actuales.

Aunque existen diversos métodos que también tendrían cabida en esta parte, como los comentados anteriormente en este apartado, se consideran demasiado complejos de desarrollar en esta aplicación, ya que los datos a almacenar no son muy complejos y tampoco muy extensos, y LocalStorage provee de las funcionalidades necesarias que permiten el correcto almacenamiento de la información en la aplicación.

Drag and Drop

¿Qué es Drag and Drop?

Es una alteración del DOM de una página Web, su objetivo es crear objetos en la Web que permitan al usuario arrastrarlos de un origen y depositarlos en un destino con la característica de que permanezcan en su destino hasta un próximo arrastre.

La incorporación de esta especificación de JavaScript en este proyecto tiene que ver con la organización, en él se pretende que el usuario busque la información que desee sobre uno o varios temas y ante la gran cantidad de la misma que encontrará, pueda separarla, ordenarla y clasificarla como desee de una forma fácil e intuitiva.

Funcionamiento

El funcionamiento básico de este mecanismo [19] consiste en localizar el cursor del ratón, reconocer cuando el usuario está haciendo clic en un elemento y de qué elemento se trata, y por ultimo mover dicho elemento.

1. Obteniendo las coordenadas del ratón, tenemos la posición del cursor. Para implementar esto en las aplicaciones Web se realiza lo siguiente:

```
document.onmousemove = mouseMove;
function mouseMove (ev) {
        ev = ev || window.event; //se obtiene el evento de
window.event
        var MousePos = mouseCoords (ev);
 }

function mouseCoords (ev) {
        if(ev.pageX || ev.pageY) {
        return {x: ev.pageX, y: ev.pageY}; //devuelve las
posiciones de x e y del ratón
        }
}
```

2. Identificar cuando se pulsa en un objeto, para lo que nos apoyamos en las funciones 'onmousedown' y 'onmouseop' y el siguiente código:

```
document.onmouseup = mouseUp;
var ObjectoDrag = null; //se crea el elemento que se pulsa
function makeClickable (objet) {
  object.onmousedown = function () {
  ObjectoDrag = this; //obtiene el objeto a arrastrar
```

```
  }
}
function  mouseUp (ev) {//se inicializa cuando se deja de pulsar
ObjectoDrag = null;
}
```

3. Por último se produce el traslado del objeto, donde lo primero es poner las posiciones en valor absoluto, es decir, ajustar las medidas en relación al margen superior izquierdo de la página Web y, como los movimientos del ratón también están relacionados con el margen superior izquierdo, esto nos marca el camino a seguir.

   Hecho esto solo nos queda cambiar la posición de superior o izquierda de la página de partida y el objeto se mueve.

   A continuación se muestra el código de la implementación de este último paso:

```
document.onmousemove = mouseMove;
document.onmouseup   = mouseUp;
var dragObject = null;
var mouseOffset = null;
function getMouseOffset(target, ev){
   ev = ev || window.event;
   var docPos = getPosition(target);
   var mousePos = mouseCoords(ev);
   return {x:mousePos.x - docPos.x, y:mousePos.y - docPos.y};
}
function getPosition(e){
   var left = 0;
   var top = 0;
   while (e.offsetParent){
         left += e.offsetLeft;
         top += e.offsetTop;
         e = e.offsetParent;
   }
   left += e.offsetLeft;
   top += e.offsetTop;
   return {x:left, y:top};
}
function mouseMove(ev){
   ev = ev || window.event;
   var mousePos = mouseCoords(ev);
   if(dragObject){
         dragObject.style.position = 'absolute';
         dragObject.style.top = mousePos.y - mouseOffset.y;
         dragObject.style.left = mousePos.x - mouseOffset.x;
```

```
            return false;
        }
    }
    function mouseUp(){
        dragObject = null;
    }
    function makeDraggable(item){
        if(!item) return;
        item.onmousedown = function(ev){
            dragObject = this;
            mouseOffset = getMouseOffset(this, ev);
            return false;
        }
    }
```

2.4 API de Twitter

Las siglas API en informática significan "Interfaz de programación de aplicaciones", que como su nombre indica ofrecen interfaces que facilitan la programación de aplicaciones y la transferencia de datos entre sistemas.

Este proyecto tiene su base en Twitter, una red social que ofrece la posibilidad al usuario de que sus seguidores (llamados "*followers*", en inglés) conozcan lo que está haciendo, y todo ello en mensajes con un máximo de 140 caracteres llamados *'tweets'*, aunque estos tweets se suelen usar para transmitir cualquier tipo de información. Por ello y aunque existen muchos y diferentes APIs, el proyecto se basará en el API de Twitter, que a su vez proporciona tres tipos de APIs: REST, Streaming y Search.

A continuación se explicarán cada uno y se justificará la elección de uno de ellos en la realización del proyecto, pero antes veamos el flujo básico de Twitter:

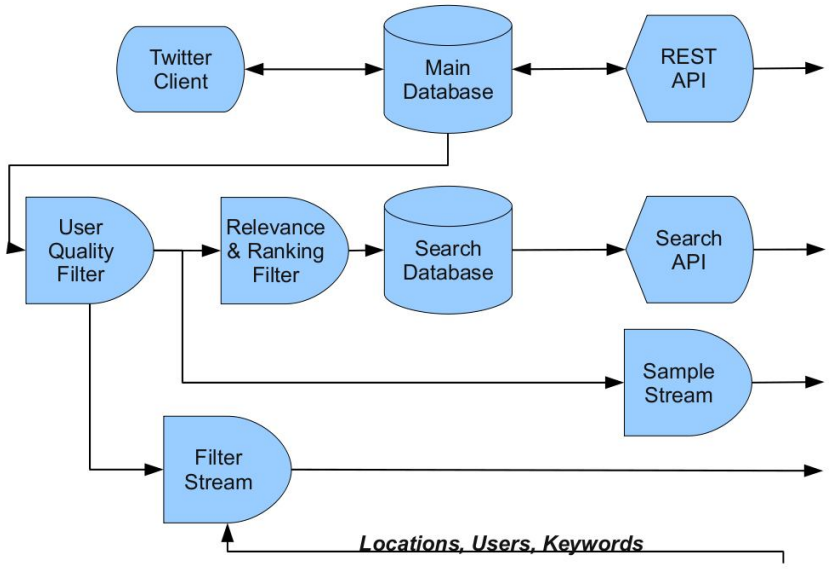


Figura 2.2. Representación del flujo de información de Twitter.

Para entender un poco mejor el flujo de tweets a continuación se explica más en detalle la imagen [20] anterior:

Los elementos que se pueden observar en la imagen se corresponden con las bases de datos de Twitter (*Main Database* y *Search Database*), los filtros por los que puede pasar un tweet (*User Quality Filter*, *Relevance & Ranking Filter* y *Stream Filter*) y por último las API's de Twitter (*REST API*, *SEARCH API* y *Sample Stream*, que nos lleva a la *STREAMING API*).

Cuando un usuario de Twitter publica un tweet se le asigna un identificador formado por la fecha y hora de publicación, el usuario que lo envió y al que es enviado si es el caso, y si el usuario permite que se conozca su ubicación ('geotagging'). Existen otros meta-datos que se adjuntan a cada tweet que contiene otro tipo de datos como hipervínculos, imágenes adjuntas, entre otras.

Después de esta identificación, el tweet entra en la base de datos de Twitter, pasa por un filtro de calidad de usuario (*User Quality Filter*) que descarta a los usuarios menos relevantes y si supera este filtro es enviado al filtro de Importancia y clasificación (*Relevance & Ranking Filter*) y por último si pasa este filtro se envía a la base de datos de búsqueda para que se añada al índice de búsqueda.

2.4.1 REST API

Twitter no solo es una red social, sino que ofrece a los desarrolladores gran variedad de servicios para permitir la automatización de la funcionalidad de Twitter. Se basa en el protocolo REST [17](Transferencia de estado representacional) que es un diseño de acoplamiento flexible de aplicaciones Web que permite a los desarrolladores acceder a la información y a los recursos mediante una invocación HTTP. Esto quiere decir que el usuario tiene la posibilidad de obtener información concreta de dominio, con solo señalar la dirección URL del sitio concreto.

El funcionamiento de REST API [21] es muy sencillo, cuando accedemos a esta API, y realizamos una búsqueda, dicha consulta se ejecuta en la base de datos principal de Twitter, permitiendo leer, crear, actualizar e incluso eliminar los tweets.

REST se basa en recursos y no en mensajes, lo que facilita el acoplamiento en el diseño de aplicaciones, además tiene un beneficio añadido y es que los recursos pueden ser cualquier cosa y el formato de los mismos puede variar.

Su simplicidad la hace altamente escalable, la única dificultad está en decidir los recursos sobre los cuales trabajar.

[17] https://dev.twitter.com/docs/api

Recursos

Como se ha comentado en el apartado anterior, REST API provee muchos recursos que proporcionan interfaces sencillas para las funcionalidades de Twitter, los recursos de los que dispone en las funcionalidades de Twitter son:

- **Plazos** → son conjuntos de tweets ordenados por antigüedad empezando por el más reciente.
- **Tweets** → los componentes básicos de Twitter que permiten una redacción de 140 caracteres.
- **Búsqueda** → busca tweets en la base de datos a partir de consultas que realiza el usuario.
- **Mensajes directos** → permite el envío de mensajes privados cortos entre los usuarios.
- **Amigos y seguidores** → los usuarios pueden seguir los intereses de otros usuarios.
- **Usuarios sugeridos** → es la organización por categorías de los usuarios que otros usuarios pueden estar interesados en seguir, podría decirse que son los más populares.
- **Favoritos** → los usuarios pueden marcar los tweets que les parezcan más importantes o interesantes y conservarlos para consultas futuras.
- **Listas** → son conjuntos de tweets de una lista de usuarios previamente seleccionados.
- **Cuentas** → se permite realizar ajustes de configuración en las cuentas de los usuarios.
- **Notificación** → permite notificaciones al usuario vía mensajes al móvil.
- **Búsquedas guardadas** → se permite a los usuarios guardar referencias de búsquedas para su posterior acceso.
- **Tendencias locales** → tendencias de contenidos específicos para determinadas áreas geográficas.
- **Lugares & geo** → permite conectar los datos de localización y tweets de los usuarios de todo el mundo.
- **'Trends'** → permite obtener el listado de los temas más populares.
- **Bloqueo** → permite bloquear y desbloquear a usuarios.
- **Información de spam** → sirve para informar al usuario de correo no deseado.
- **OAuth** → se utiliza para la autenticación OAuth.

- **Ayuda** → ofrece soporte en la utilización de Twitter a los usuarios.
- **Legal** → tiene métodos para abordar los temas jurídicos en las aplicaciones.
- **Obsoleto** → detecta los métodos que no se usan y con el tiempo son eliminados de la plataforma.

Construcción de un servicio Web REST

Para la construcción de una REST API, que como ya se ha comentado se sirve de invocaciones HTTP, es necesario la utilización e identificación de cada una de las tareas que se desean automatizar con la REST API con cada una de las siguientes peticiones del protocolo HTTP, a continuación se muestra una tabla obtenida de 'IBM developerWorks [22]', que las relaciona:

Mapeo CRUD/HTTP	
Tareas	**Petición HTTP**
Create	POST
Read	GET
Update	PUT
Delete	DELETE

Tabla 2.2. Mapeo CRUD/HTTP.

Con esto, tenemos la parte de la funcionalidad pero los recursos también deben ser construidos aunque como ya se ha comentado con anterioridad el formato de los mismos no es relevante.

Estos recursos deben ser los suficientes para poder utilizar la funcionalidad del servicio en su totalidad. Por poner un ejemplo, si queremos un servicio Web REST de apuestas futbolísticas necesitaremos los equipos, los apostantes y las apuestas como recursos a crear.

En la implementación, la sintaxis para la creación de una apuesta en XML podría ser:

```
<apuesta nombre="Apuesta1" fecha="01-07-2011" cantidad="100"
id="1">
 <uri>/apuesta/1</uri>
</apuesta>
```

Por otro lado, los elementos tienen que estar asociados unos con otros.

Continuando con el ejemplo anterior, tenemos que un apostante puede realizar una o varias apuestas y por tanto son recursos que deben estar relacionados, veamos como se implementaría:

```
<apostante nombre="Apostante1" DNI="DNI1”>
 <uri>apostante/DNI1</uri>
 <description> Descripción del apostantes y sus
apuestas</description>
 <apuestas>
  <apuesta nombre="Apuesta1" fecha="01-07-2011" cantidad="100"
id="1">
  <uri>/apuesta/1</uri>
  </apuesta>
  <apuesta nombre="Apuesta2" fecha="01-07-2011" cantidad="55"
id="2">
  <uri>/apuesta/2</uri>
  </apuesta>
 </apuestas>
</apostante>
```

Conclusiones

El uso de REST API de Twitter podría ser una buena elección puesto que permite automatizar la gran mayoría de las funcionalidades de Twitter de forma manual y por tanto ofrece al usuario gran comodidad en la navegación y obtención de la información deseada, por ejemplo, a la hora de obtener datos concretos de un usuario o filtrar los tweets en base a algún criterio especifico y recogerlos en su blog, etc.

En definitiva con la Interfaz de programación de aplicaciones REST se cubren las funcionalidades básicas de Twitter, como leer los mensajes directos, seguir y dejar a los usuarios, enviar mensajes directos, manipular listas, etc. Pero aun así no se ajusta completamente a lo que se busca para la realización de este proyecto ya que se espera que el número de peticiones HTTP sea elevado y esto dificultaría la implementación de esta API en la aplicación. Además REST API coge la información directamente de la base de datos de Twitter (ver figura 2.2) y lo que se desea es que los tweets hayan pasado previamente por el filtro de búsqueda.

Es un API muy completo pero lo que se busca en este proyecto es implementar búsquedas de tweets ya publicados y esto se puede llevar a cabo de una manera más sencilla que se describirá en apartados posteriores.

2.4.2 Streaming API

Streaming API [23] permite realizar consultas a través de peticiones HTTP de sólo lectura, es decir, que cuando se realiza cualquier búsqueda los resultados de la misma no se pueden modificar ni eliminar.

Al ejecutar una búsqueda en esta API, se produce una conexión permanente entre el usuario y los servidores de Twitter que hace que la información fluya de manera continua, permitiendo la búsqueda de tweets casi en tiempo real y sin necesidad de actualizar las búsquedas, simplemente los tweets se van mostrando según se van publicando.

Como en este tipo de API el flujo de información es continuo el único filtro que puede aplicarse es un filtrado por palabras clave o usuarios y por tanto la información que se obtendrá a través de sus consultas será mucho mayor que en el resto de API's.

Esta API de Twitter es la que más rendimiento ofrece a los usuarios en la obtención de la información en tiempo real. En ella existen tres productos principales en los que no se profundizará debido al descarte de la implementación de esta API en este proyecto, que son:

- **Streaming API [24]** → se basa en los estados de los usuarios y filtra la información de diversas maneras, por el identificador del usuario, por localización, por claves, etc.
- **Users stream** → tiene que ver con los datos necesarios para la actualización de la interfaz del usuario y tiene el objetivo de proporcionar actualizaciones a los clientes de Twitter.
- **Site streams** → tiene el objetivo de permitir que se abran varias conexiones desde el mismo host o servicio, normalmente el servicio son sitios Web.

Conclusiones

La realización de este proyecto no se ha pensado para grandes cantidades de información por lo que la implementación de esta API se descarta. Aunque es importante destacar de ella la velocidad con la que se transfiere la información, que a pesar de puede variar en función del ancho de banda del usuario y del servidor y de la sobrecarga del mismo, siempre será mas rápida que el resto de API's.

Pero las ventajas de poder obtener la información de una manera más rápida y sin necesidad de estar actualizando la búsqueda, no compensan en este proyecto si se piensa en la complejidad de su implementación, puesto que dentro de los objetivos fijados no es relevante la velocidad de obtención de la información. Además el resto de API's ofrecen un tiempo de respuesta razonable.

2.4.3 Search API

La Search API [25] de Twitter permite a los usuarios integrar los resultados de las búsquedas en Twitter a sus propias aplicaciones, lectores y clientes.

Esta API realiza las consultas de tweets en la base de datos de búsqueda. Dicha base de datos es solo de lectura por lo que no permitirá crear, eliminar o modificar ninguno de los tweets resultantes de la búsqueda.

El método para construir una consulta en la Search API de Twitter consiste en generar una URL que represente dicha consulta. La base para la creación de esta URL se centra en tres aspectos:

1. Primero existe una parte fija: http://search.twitter.com/search.
2. Después se añade el formato en que se desean recibir los datos (atom, json, etc.) con esta estructura: json?q=
3. Y Por ultimo, se añaden los datos de la consulta a partir de los operadores que se muestran en la tabla 2.3.

Los operadores que permiten al usuario modificar el comportamiento de la búsqueda son:

Operador	***Busca Tweets que contengan:***
Dos Palabras	Ambas palabras, “Dos” y “Palabras”.
“Cadena”	La cadena exacta.
#Palabra	El *hashtag* “Palabra”
X OR Y	“X” o “y”
From: X	Enviado por el usuario @X
To: Y	Se envía al usuario @Y
Place:opentable:2	Sobre el lugar OpenTable con id=2
Place: 11111	Sobre el lugar con ID 11111
@nombre	Menciones al usuario @nombre
cadena since: año-mes-dia	La cadena “cadena” y fue enviado dese la fecha indicada.

Cadena until:año-mes-dia	La cadena "cadena" y fue enviado antes de la fecha indicada.
Movie-scary ;)	"movie" pero no "scary" y con actitud positiva
Trabajando :(	"Trabajando" y con actitud negativa
De viaje?	Haciendo una pregunta relacionada con viajes
Perro filter:links	Las URL que contengan la palabra "Perro"
Palabras source:tweet_button	Tweets que contengan "Palabras" y hayan entrado a través del botón "Tweet" en una página Web

Tabla 2.3. Operadores para la realización de búsquedas.

Todos estos operadores se pueden combinar para formar consultas más específicas y poder filtrar la información no deseada.

Un ejemplo en el que se genera una URL para una consulta en la que se desean recibir los tweets que contengan la palabra 'uc3m' en formato 'json' se implementaría de la siguiente manera:

http://search.twitter.com/search.json?q=uc3m

Esta API ofrece dos posibilidades similares de uso, los formatos Atom y JSON, veamos que en que consiste cada una de ellas:

ATOM API:

Con el uso de esta API el usuario puede obtener los resultados de una búsqueda como un 'feed Atom estándar[18]'. Este 'feed Atom' es un medio de redifusión de contenido Web que se utiliza para suministrar información actualizada de manera automática.

Por otro lado con ATOM API [26] también se permite crear una URL a la hora de realizar una búsqueda en los tweets, esto se puede llevar a cabo fácilmente añadiendo la codificación URL de la búsqueda a la dirección del servicio Atom, que es, *http://search.twitter.com/search.**atom**?q = <query>.*

Algunas de las consultas (<query>) realizadas en este formato pueden ser para buscar los tweets que contengan alguna referencia a un usuario concreto o que contengan una palabra determinada, o buscar tweets de un usuario, entre otros.

Pero obtener la URL del 'feed Atom' de una consulta puede realizarse más fácilmente si se realiza a través de Twitter, haciendo clic en el 'feed' de la consulta en

[18] http://www.ibm.com/developerworks/opensource/library/x-phpatomfeed/index.html

el menú lateral o utilizando la búsqueda avanzada.

JSON API:

Esta API (JSON [27]) es muy similar a la API ATOM, la única salvedad es que ésta obtiene la información en formato JSON.

Es posible crear una URL para las búsquedas de manera similar a ATOM con *http://search.twitter.com/search.**json**?q = <query>*. Además esta API es compatible con el parámetro de devolución de llamada que devuelve los resultados utilizando el formato JSONP [28].

Esta API es un índice de los tweets, pero en el solo se tienen en cuenta los creados desde el mismo momento de la consulta hasta un máximo de 9 días atrás, lo que limita al usuario a no encontrar tweets de más de una semana.

Es importante que las búsquedas no sean demasiado complejas de lo contrario la búsqueda genera un error y no se puede llevar a cabo. Y otra característica que es importante en esta API es que todas las búsquedas son anónimas.

Conclusiones

Una vez se han analizado las características que ofrece esta API, se puede concluir que dispone de unas funcionalidades que posibilitan las búsquedas de una manera sencilla. Con solo seguir las pautas mencionadas en el sub-apartado anterior se pueden generar URL's que representen consultas tan complejas como se desee, y obtener los mismos resultados que con el uso de las demás API's.

Principalmente, por esta razón se ha decidido llevar a cabo la implementación de esta API, para satisfacer el objetivo de seleccionar la información de Twitter y poder hacerlo de la manera más fácil y eficaz posible.

2.4.4 Conclusión

Tras conocer las funcionalidades y la descripción de cada una de las API que ofrece Twitter, se ha llegado a la conclusión de que la API que mejor se ajusta a las necesidades y objetivos de este proyecto es la Search API.

Por un lado, se ha descartado usar REST API porque con ella tenemos la limitación del filtrado de la información, lo que para los futuros usuarios de esta aplicación Web es necesario, puesto que una de las funcionalidades es filtrar la información de acuerdo a una búsqueda realizada por el usuario y dar la posibilidad de clasificarla.

Por otro lado, no contamos con la Streaming API debido a la gran cantidad de datos que ofrece, esta API esta destinada para procesar grandes cantidades de información y la aplicación final de este proyecto no busca información a grande escala. Además, otra limitación que tiene Streaming y que no se acopla al proyecto es que realiza filtrado automático y en la aplicación Web lo que se busca es que sea el usuario quien decida qué hacer con los resultados de la búsqueda.

Por todo esto y porque nos permite realizar búsquedas concretas y eficaces, se implementará para la realización de este proyecto Search API.

Capítulo 3

Desarrollo de la solución

3. DESARROLLO DE LA SOLUCIÓN

El método que se utilizará para llevar a cabo un sistema que permita cumplir con los objetivos comentados, es la realización de una aplicación Web que permita al usuario interactuar y realizar clasificaciones de la información que se obtiene en búsquedas en la red social Twitter.

Esta aplicación se ha desarrollado principalmente en HTML 5 y dado forma a su estilo con CSS 3, pero además para llevar a cabo la funcionalidad necesaria se han utilizado las tecnologías PHP y JavaScript. Tecnologías descritas más en detalle en el capítulo 2 (Estado del arte).

Los principales motivos por los que se ha decidido utilizar HTML 5 para el diseño de esta aplicación Web son:

1. Este estándar permite crear una estructura más ajustada al contenido de la página, incluyendo etiquetas específicas para cada parte. Esto hará que el mantenimiento y actualización sea más sencillo e intuitivo para el desarrollador y facilitará la inclusión de mejoras en el futuro.
2. Además utiliza etiquetas semánticas, es decir, etiquetas que contienen meta-información acerca del elemento o elementos que se encuentran dentro de esa etiqueta, consiguiendo que la estructura de la aplicación tenga un mayor significado y por tanto tenga más sentido.
3. La descarga de aplicaciones Web son más rápidas, es decir, con HTML 5 se cargan primero los elementos de la página más importante y los que tardan menos en cargarse, dejando los demás para después. Esto es muy beneficioso porque el usuario puede ir teniendo acceso a la Web mientras termina de cargarse y por tanto evita que el usuario la abandone por esperar demasiado tiempo a que se cargue.

Además, esta nueva tecnología ofrece dos ventajas más que serían importantes de cara a mejorar la aplicación:

1. las nuevas etiquetas <audio> y <video> son una importante aportación en HTML 5 ya que permiten la incrustación de videos y audio sin necesidad de embeber un reproductor multimedia, como se solía hacer en versiones anteriores. Esta característica podría ser de mucha utilidad, por ejemplo si se quisiera ampliar la aplicación e incluir en la información de los tweets a clasificar los videos y/o audios que publica cada usuario.
2. De la misma manera se podría actualizar la apariencia de la aplicación y añadir formas en 2D, para lo cual HTML 5 dispone de la etiqueta <canvas>

que nos permite lo comentado sin necesidad de tener que incluir plataformas multimedia que complicaban mucho más el trabajo.

Por otro lado, la aplicación se ha desarrollado en una plataforma Windows 7 y almacenado en el servidor virtual 'Xampp'[29], aunque en lugar de esta opción podría haber utilizado otro Sistema Operativo, por ejemplo, Linux y un servidor Web cualquiera que soporte PHP.

De acuerdo con las tecnologías disponibles relacionadas con este proyecto que existen actualmente y que fueron analizadas para la realización de la aplicación, el desarrollo del proyecto se lleva a cabo de acuerdo a los siguientes apartados:

- **API de Twitter** → el objetivo principal del proyecto es permitir la clasificación de la información de la red social Twitter, pero no implementa un buscador nuevo sino que se sirve de las Interfaces que ofrece Twitter a los desarrolladores para acceder a la información de su base de datos.
- **Tratamiento y presentación de la información** → la información que se obtiene como resultado de la búsqueda en Twitter es recibida en formato JSON por lo que se realiza una conversión y tratamiento de la misma para que pueda ser mostrada de una manera cómoda para el usuario.
- **Clasificación de la información** → la clasificación es el objetivo primordial de esta aplicación, para ello se han creado dos listas de clasificación que permiten mediante el mecanismo 'Drag and Drop' posicionar el conjunto de información de cada 'tweet' en las distintas zonas.
- **Almacenamiento de la información** → es de vital importancia que el usuario no pierda la información clasificada, por este motivo se utiliza el mecanismo de 'LocalStorage' (apartado 2.3.3) de JavaScript para guardar la información procesada. Con este método el usuario podrá disponer de la información aunque realice otra búsqueda o recargue la página.
- **Personalización de la clasificación** → se permite una ligera personalización de la clasificación ofreciendo al usuario la posibilidad de asignarle un título a la clasificación si lo desea.
- **Descarte de la información** → es igual de importante para filtrar la información, almacenarla y permitir su descarte. Esta funcionalidad ha sido implementada mediante el uso de botones que se encargan del borrado de la información contenida en una zona de clasificación o en las dos y además se puede eliminar cada 'tweet' moviéndola a la papelera.
- **Descarga del contenido** → se ofrece un almacenamiento del contenido clasificado mediante la descarga de un fichero en formato '.pdf' que contendrá la información contenida en la zona de clasificación que se desee descargar.

Para llevar a cabo una solución que englobe todo lo anteriormente mencionado se ha creado un prototipo de una aplicación Web que cumple con la funcionalidad y los objetivos descritos.

A continuación se resumen las fases que han sido necesarias llevar a cabo para el desarrollo de este prototipo:

- **Análisis** → recaudación de la información sobre las técnicas que serán de utilidad para llevar a cabo algún aspecto de la aplicación Web.
- **Diseño** → elección de los métodos y técnicas que se aplicarán para llevar a cabo la aplicación.
- **Implementación** → desarrollo del prototipo que cumpla con las necesidades y objetivos fijados.
- **Evaluación** → realización de las pruebas para la comprobación del correcto funcionamiento de la misma y solución de los errores encontrados.

3.1 Análisis

Una investigación de 'comScore' [30], una empresa líder en la medición del mundo digital, realizó un artículo en Abril de 2011 sobre la influencia y uso de Twitter en los distintos países. En el estudio se presentan algunos de los primeros países donde Twitter tiene un mayor alcance ofreciendo dicha información en un 'ranking' de países donde Holanda es el país donde mayor número de habitantes usa Twitter. Veamos dicha información:

1. Países Bajos → con un 26,8%
2. Japón → con un 26,6%
3. Brasil → con un 23,7%
4. Indonesia → con un 22.0%
5. Venezuela → con un 21,0%
6. Canadá → con un 18,0%
7. Argentina → con un 18,0%
8. Turquía → con un 16,6%
9. Filipinas → con un 16,1%
10. Singapur → con un 16,0%

Estos países son aquellos donde Twitter tiene un mayor alcance, pero además de ellos Twitter está presente en muchos más países como Estados Unidos y Europa.

Con estos datos se puede decir que Twitter es una de las redes sociales más populares en el mundo entero y por tanto la aplicación que se desarrolla en la realización de este proyecto tendrá mucha más cabida en la Web.

Además un artículo que habla sobre el flujo de información de la red social Twitter, publicado en Marzo de 2011 en 'MuyInteresante.es' desvela que *'Se publican 110 millones de mensajes cada día y que se difunden entre alrededor de 200 millones de usuarios*.' Esta información es de vital importancia, puesto que con ella se obtienen datos recientes sobre el aumento de la información en la Web y la existencia de un verdadero problema con el tratamiento de tanta información.

Twitter es una gran red social pero en ella solo podemos consultar la información. Es cierto que ofrece la posibilidad de agregar información en un apartado en el que se pueden crear listas y agregar usuario a esas listas, pero en ella aparecen todos los 'tweets' que han insertado recientemente todos los usuarios que se encuentren añadidos a dicha lista, lo que no resulta de mucha utilidad si solo se está interesado en una información concreta de algún usuario. Además, como se ha comentado sólo se puede consultar dentro de la red social, es decir, no cuenta con una funcionalidad que permita al usuario desde su cuenta Twitter, volcar o transferir la información, por ejemplo en ficheros, mas allá del entorno Twitter.

Todas estas "carencias" en las funcionalidades de Twitter ofrecen grandes posibilidades a este proyecto, que se centrará en permitir la clasificación de los tweets de manera individual hasta en dos categorías distintas, guardando solo que contengan la información deseada y permitiendo que dicha información clasificada pueda ser descargada y consultada también fuera del entorno de la aplicación.

Una vez que se analizaron las posibilidades y el alcance de la aplicación se pasa a realizar el diseño del prototipo de la aplicación Web.

3.2 Diseño

En esta fase se pretende diseñar la arquitectura en la que se desplegará la aplicación así como el diseño de la funcionalidad especificada.

Como introducción comentar que se ha optado por un diseño sencillo, que sea intuitivo y fácil de manejar y que además permite interactuar activamente al usuario con la aplicación. Los colores elegidos (naranja y gris) en distintas tonalidades se fijan para dar una sensación de calidez y confianza al usuario (tonalidades naranja) pero a la vez seguridad y formalidad (tonalidades grises sobre fondo negro).

Además, para no saturar al usuario con información innecesaria, se ha decidido seleccionar la información que se mostrará al usuario, ya que cuando se realiza una búsqueda en la base de datos principal de Twitter, esta devuelve todos los datos

disponibles relacionados con la información a buscar, pero en esta aplicación son de relevancia: el nombre del usuario y el enlace a su perfil de Twitter, la imagen del mismo y por supuesto la información que publica. Estos datos dan el formato a los tweets.

3.2.1 Diseño arquitectónico

La arquitectura Hardware que se ha utilizado en el desarrollo de la aplicación, es una arquitectura basada en un servidor que accede a la información de la base de datos principal de Twitter. A continuación se muestra la topología de dicha arquitectura:

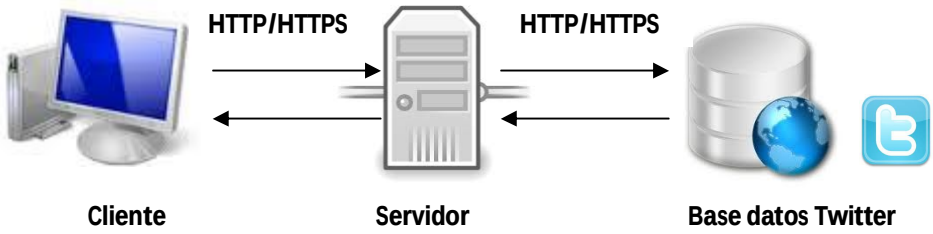


Figura 3.1. Arquitectura del servicio

En la figura anterior el cliente accede al servidor mediante peticiones HTTP.

Cuando el usuario realiza una búsqueda, el servidor lanza una operación de petición de información a la base de datos de Twitter. Esta plataforma se encarga de verificar que los datos de autenticación del usuario en Twitter son correctos y si es así devuelve la información solicitada, que es tratada internamente por la aplicación y mostrada al usuario en la misma.

Con respecto a la arquitectura Software, la aplicación se despliega en el servidor 'Xampp' donde reside la instalación de PHP y Apache. La aplicación no necesita de instalación sino que reside en el servidor Web.

Una vez planteado el diseño que tendrá la aplicación el siguiente paso es empezar con la implementación de la misma.

3.3 Implementación

En esta fase del desarrollo se detalla la funcionalidad de la que dispone la aplicación y como se ha implementado en la misma.

3.3.1 Diagramas UML

Se ha utilizado un diagrama de casos de uso para conocer el flujo de eventos entre usuario y sistema, de una manera general.

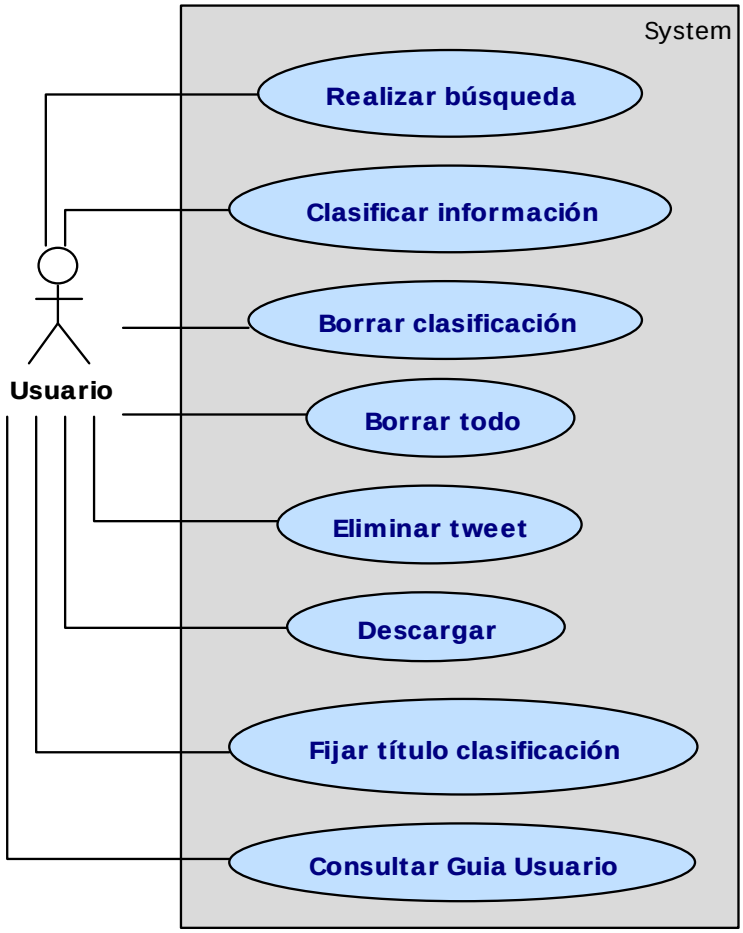


Figura 3.2. Diagrama de casos de uso de la aplicación.

Y para ver de una manera más grafica en qué consiste cada caso de uso se han utilizado los diagramas de actividad, que reflejan las actividades que se realizan para llevar a cabo las funcionalidades de la aplicación.

Realizar búsqueda:

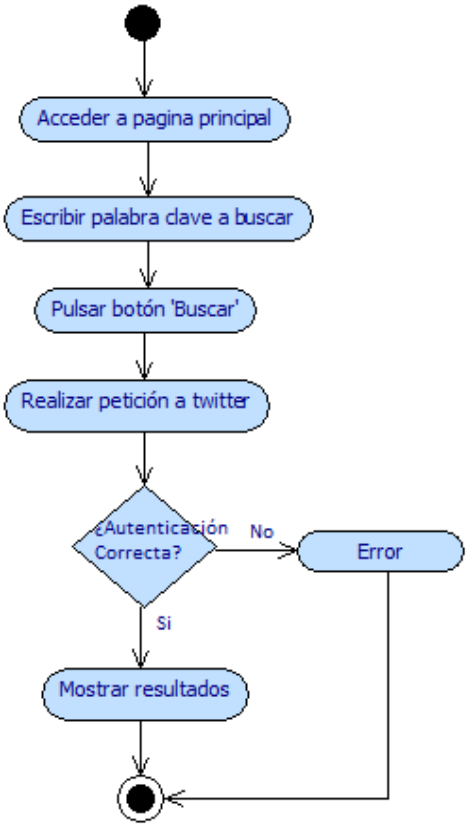


Figura 3.3. Diagrama de actividad de búsqueda.

Clasificar información:

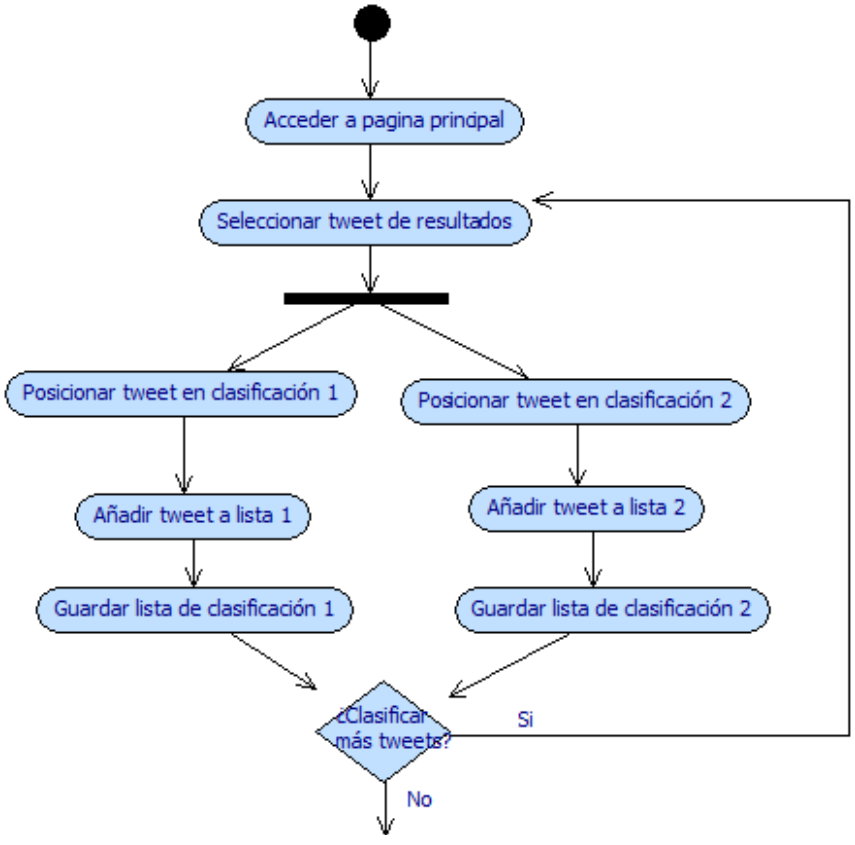


Figura 3.4. Diagrama de actividad de clasificación.

Poner título a clasificación:

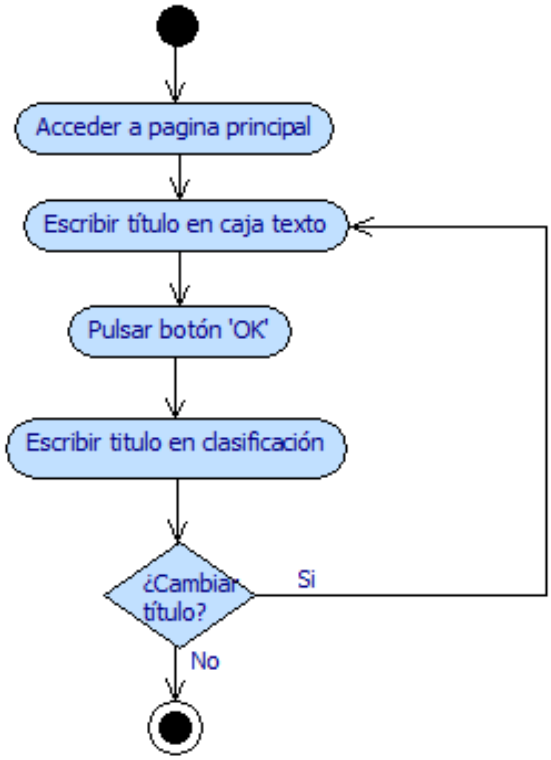


Figura 3.5. Diagrama de actividad de asignación título.

Borrar contenido de una clasificación:

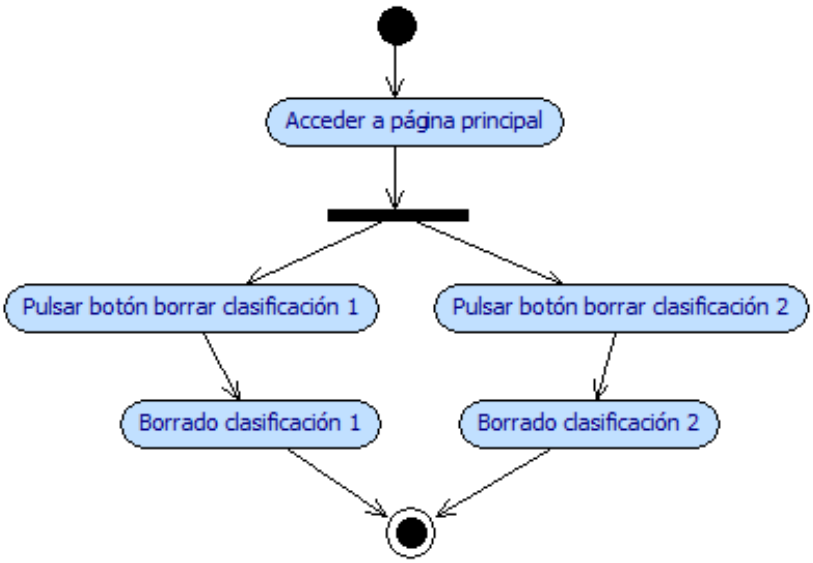


Figura 3.6. Diagrama de actividad de borrado de clasificación.

Borrar todo el contenido clasificado:

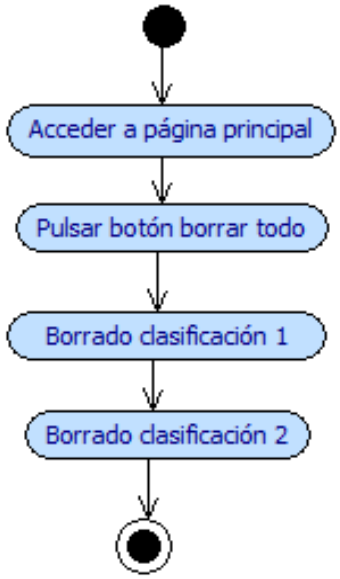


Figura 3.7. Diagrama de actividad de borrado de todas las clasificaciones.

Eliminar un tweet:

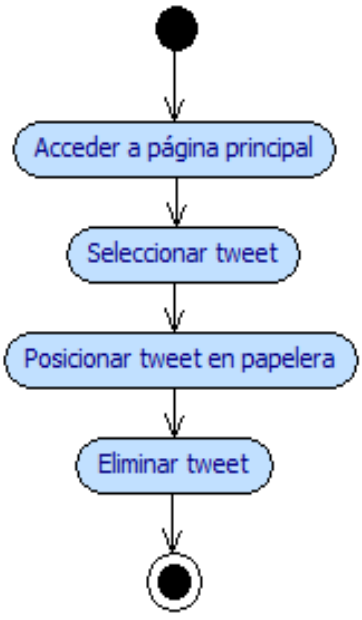


Figura 3.8. Diagrama de actividad de eliminación de un tweet.

Descargar la información de una clasificación:

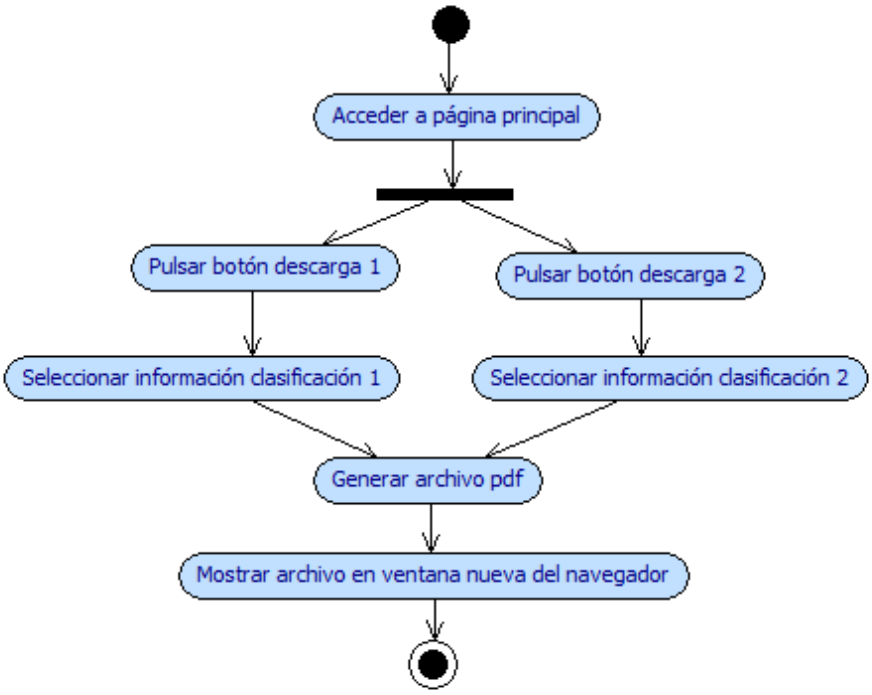


Figura 3.9. Diagrama de actividad de descarga de información de una clasificación.

Consultar guía de usuario:

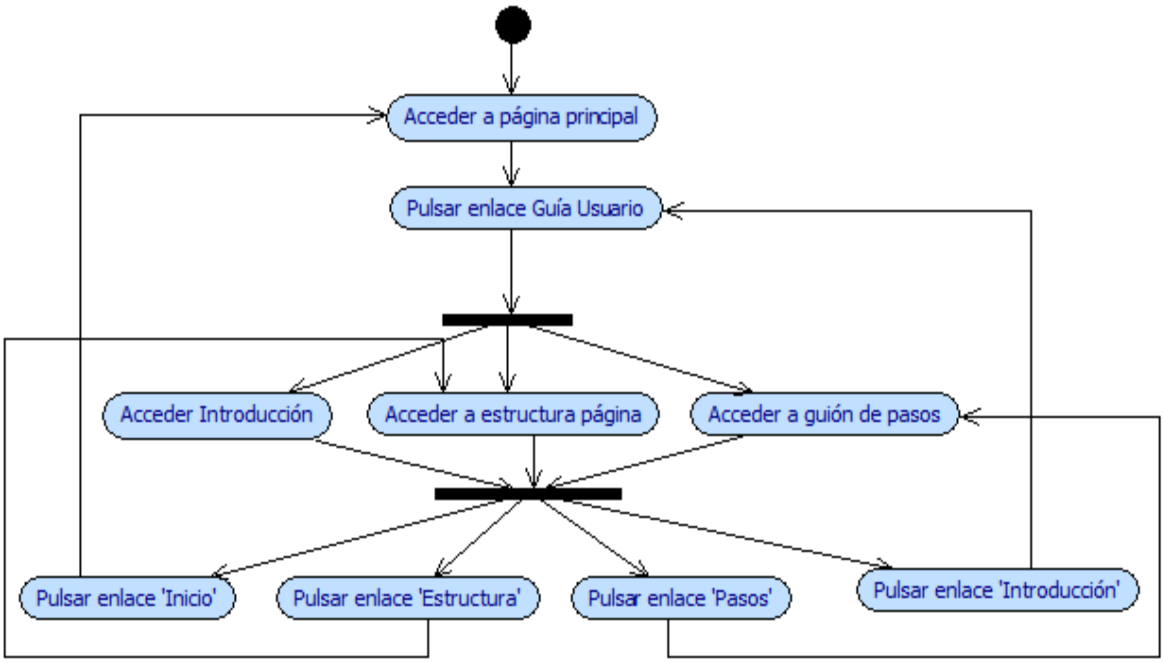


Figura 3.10. Diagrama de actividad de consulta de guía de usuario.

3.3.2 Presentación visual de la implementación

Por otro lado, se explica a través de imágenes del prototipo como se ha implementado la funcionalidad comentada:

En primer lugar, se parte de la página principal cuya estructura se ha implementado en una reciente versión del lenguaje de marcado de meta-etiquetas, HTML 5:

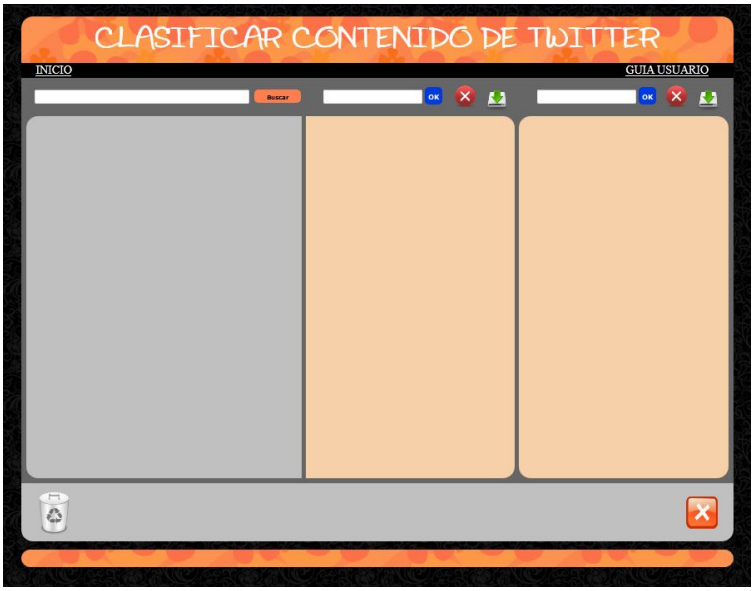


Figura 3.11. Pagina principal de la aplicación.

En ella el usuario puede realizar búsquedas de información en Twitter, con la que obtendrá los 'tweets' que contengan la palabra o palabras que se introduzcan en la búsqueda. El acceso a la información de la red social Twitter se ha realizado a través de la 'API Search' que ofrece Twitter a los desarrolladores (Ver apartado 2.4). Esta API de Twitter es una interfaz que ofrece a los desarrolladores librerías que permiten incluir los resultados de las búsquedas en Twitter en sus propias aplicaciones Web. Estos resultados se reciben en la aplicación en formato 'JSON' y la aplicación realiza un tratamiento de los mismos seleccionando la imagen de usuario, el enlace que lleva al perfil del usuario y el contenido textual escrito por el mismo. Estos resultados se muestran en la aplicación en una lista de 'tweets', los cuales siguen el siguiente formato:

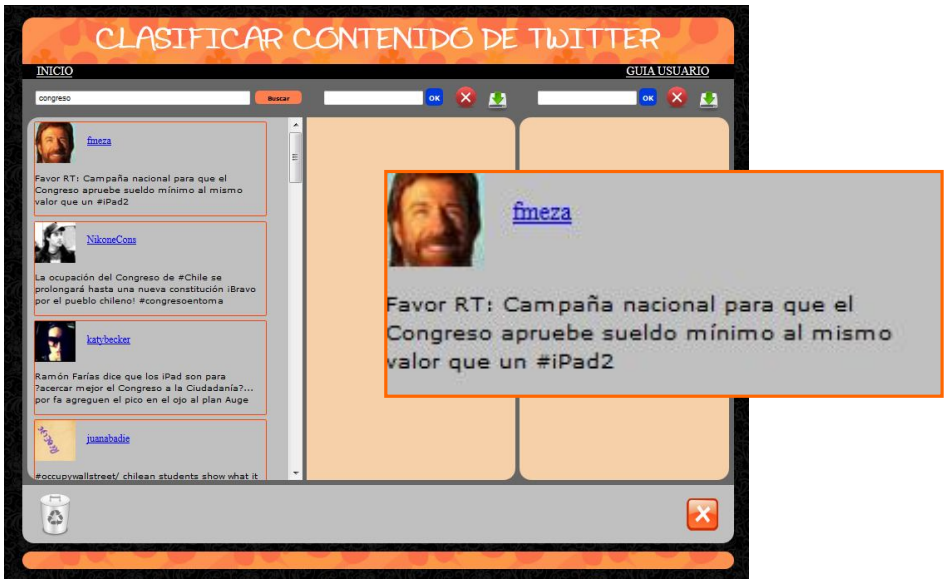


Figura 3.12. Formato de los resultados de la búsqueda.

Como se puede ver en la imagen, un 'tweet' está compuesto de un enlace, una imagen de perfil y un texto. El texto es el que contiene la información a buscar, la imagen es la imagen del usuario de Twitter que colgó la información y el enlace lleva al perfil del usuario en la red social.

Una vez que se tienen los resultados de la búsqueda la siguiente funcionalidad que ofrece la aplicación es la de clasificar los resultados para lo que se dispone de dos zonas de clasificación definidas en la estructura HTML. Esta acción se lleva a cabo por medio de la implementación de la técnica 'Drag and Drop' que hace posible mover los tweets resultantes a las zonas de clasificación.

Además, la información de estos 'tweets' se va guardando de manera automática cuando estos son arrastrados a una zona de clasificación mediante un proceso de almacenamiento local. Este almacenamiento es implementado en la aplicación usando una base de datos de cliente que se basa en pares de valores y que reside en el navegador del usuario ('localStorage' de JavaScript), consintiendo recuperar la información clasificada cuando se realice otra búsqueda o cuando se recargue la página. Esto hace posible que el usuario no pierda información y que la tenga disponible durante todo el proceso de clasificación. Veamos un ejemplo en el que se clasifican varios tweets:

Figura 3.13. Zonas de clasificación.

A partir de la información clasificada, se permite cambiar el título de la clasificación escribiendo en la caja de texto correspondiente a cada zona de clasificación. Para la implementación de esta característica se ha creado la caja de texto con un botón relacionado que cuando se introduce texto en dicha caja y se pulsa el botón, modifica el contenido de la cabecera de la zona de clasificación, para lo cual se ha utilizado Jquery que permite seleccionar los elementos de la página Web y añadir, modificar o eliminar contenido. La siguiente figura muestra dicha funcionalidad:

Figura 3.14. Caja de texto para la asignación del título.

Además pulsando en el botón eliminar se eliminan los 'tweets' clasificados en la zona correspondiente, en este caso para eliminar el contenido se ha seleccionado el elemento y usado la propiedad '*innerHTML*' que guarda el texto o elementos contenidos en una parte de la estructura. Llamando a una función que ponga a vacío el contenido de la propiedad para la lista de clasificación se consigue eliminar los 'tweets'. Después se eliminan del navegador con la función 'removeItem' de LocalStorage.

Figura 3.15. Botón de eliminación.

Y de la misma manera, pulsando el botón de eliminar todo se elimina todo el contenido clasificado hasta el momento.

Figura 3.16. Botón de eliminación del contenido de las clasificaciones.

También se puede eliminar un 'tweet' arrastrándole a la papelera, funcionalidad que se consigue también con 'Drag and Drop' y el atributo 'hide' que oculta la información contenida en la papelera.

Figura 3.17. Eliminación de un tweet.

La aplicación ofrece la posibilidad de descargar el contenido de las zonas de clasificación (por separado) pulsando el botón de descarga. Para poder conseguir esta característica se ha creado una aplicación paralela implementada en PHP que se encarga de generar el fichero '.pdf' con el contenido. Este contenido es enviado desde la aplicación principal, por medio de un formulario a través del método 'POST', más tarde, la aplicación de descarga recibe los datos, los procesa, genera un '.pdf' y lo muestra en una ventana nueva del navegador para que el usuario lo guarde si así lo desea. Siguiendo el ejemplo anterior, los ficheros con el contenido descargado quedarían así:

Figura 3.18. Creación de ficheros de contenido.

Por último, se presenta una guía de ayuda al usuario que explica de forma resumida lo que se puede llevar a cabo en la aplicación, la estructura de la página y los pasos necesarios para realizar una búsqueda y clasificar información.

Figura 3.19. Guía de usuario.

Ya se tiene disponible la aplicación Web por lo que se pasa a comprobar su correcto funcionamiento.

3.4 Validación

En esta fase del desarrollo se describe un caso ejemplo de un usuario que está interesado en buscar información sobre el referéndum del 15 de Octubre de 2011 y a partir del cual se verificará el correcto funcionamiento de la aplicación.

El movimiento producido el 15 de Octubre de 2011 trajo consigo una gran cantidad de información. Por un lado se publicó información explicativa sobre su finalidad y objetivos para aquellas personas que estuvieran interesadas en formar parte del referéndum, por otro lado los usuarios de Twitter fueron publicando información acerca de lo que ocurría durante la manifestación y más tarde se publicaron tweets con comentarios sobre distintos aspectos de la experiencia. Además a fecha de hoy todavía se sigue publicando información, por ejemplo, acerca de los primeros efectos que produjo en política.

Caso ejemplo

Se supone que un usuario está interesado en recolectar y almacenar alguna información sobre las manifestaciones que tuvieron lugar el 15 de Octubre de 2011.

Este usuario entra en la aplicación y comienza a buscar información relacionada con dicho tema.

Ha decidido clasificar la información que encuentre en dos categorías: las manifestaciones que se produjeron en Madrid y las que tuvieron lugar en Barcelona. Comienza su búsqueda introduciendo la palabra 'Madrid' esperando encontrar información sobre las manifestaciones que se celebraron en Madrid, pero los resultados que se obtienen son de gran diversidad y se relacionan mucho con el equipo de fútbol Real Madrid.

Figura 3.20. Resultados de la búsqueda 'Madrid'.

Tal y como se describió en el apartado 2.4.3 las peticiones de búsquedas a través de la Search API de Twitter, se realizan a través de la generación de URL's que representan las consultas, pues bien, para este caso la URL asociada a la consulta 'Madrid' es:

```
http://search.twitter.com/search.json?q=Madrid
```

Esta URL devuelve los resultados en formato JSON, es decir, devuelve un arreglo que contiene todos los datos de los objetos (tweets) relacionados con la información buscada. Estos datos llegan a la aplicación de la siguiente manera:

```
{"completed_in":0.217,"max_id":127767953570676737,"max_id_str":"12776
7953570676737","next_page":"?page=2&max_id=127767953570676737&q=Madri
d","page":1,"query":"Madrid","refresh_url":"?since_id=127767953570676
737&q=Madrid","results":[{"created_at":"Sat, 22 Oct 2011 15:27:09
+0000","from_user":"igotjersey","from_user_id":410628813,"from_user_i
d_str":"410628813","geo":null,"id":127767953570676737,"id_str":"12776
7953570676737","iso_language_code":"en","metadata":{"result_type":"re
cent"},"profile_image_url":"http://a2.twimg.com/profile_images/156810
0906/image_normal.jpg","source":"&lt;a
href=&quot;http://twitter.com/&quot;&gt;web&lt;/a&gt;","text":"ready
stok 200idr free ongkir madrid home M kw ori
http://t.co/PUw6XLxP","to_user_id":null,"to_user_id_str":null},{"crea
ted_at":"Sat, 22 Oct 2011 15:27:07
+0000","from_user":"bowoistimewa","from_user_id":222490120,"from_user
_id_str":"222490120","geo":{"coordinates":[-
6.3100,106.6846],"type":"Point"},"id":127767947346325504,"id_str":"12
7767947346325504","iso_language_code":"sv","metadata":{"result_type":
"recent"},"profile_image_url":"http://a2.twimg.com/profile_images/159
7925008/328373669_normal.jpg","source":"&lt;a
href=&quot;http://ubersocial.com&quot;
rel=&quot;nofollow&quot;&gt;\u00DCberSocial&lt;/a&gt;","text":"MNCTV
Sabtu 23:30 - Liverpool vs Norwich... tvone Minggu 01:00 - Malaga vs
Madrid... tvone Minggu 03:00 Barcelona vs
Sevilla","to_user_id":null,"to_user_id_str":null},…],"results_per_pag
e":15,"since_id":0,"since_id_str":"0"}
```


Figura 3.21. Resultados de una búsqueda en JSON.

En la Figura 3.21 se muestran los resultados que se obtienen de la consulta, presentados por el arreglo de objetos 'results' (color Azulón) y dentro del cual se encuentran todos los datos de los tweets (color rojo). Pero además también se muestran datos de la consulta, como el tiempo que se tardó en recuperar la información o la consulta realizada ('Madrid') y que están representados en la imagen con el color azul marino.

Como los resultados no son los que se esperaba, el usuario decide afinar un poco más su búsqueda introduciendo 'Manifestación 15 Octubre Madrid', con esta búsqueda encuentra mucha información sobre lo que buscaba.

Figura 3.22. Resultados de la búsqueda 'Manifestación 15 Octubre Madrid'.

A continuación el usuario clasifica la información de los resultados que considera de su interés arrastrándolos a la primera zona de clasificación y le asigna el título significativo 'Referéndum 15 de Octubre en Madrid'.

Figura 3.23. Clasificación de los tweets en la categoría 'Referéndum 15 de Octubre en Madrid'.

Después realiza la búsqueda 'Manifestación 15 Octubre Barcelona' para encontrar las manifestaciones que tuvieron lugar en Barcelona. Obtiene los siguientes resultados:

Figura 3.24. Resultados de la búsqueda 'Manifestación 15 Octubre Barcelona'.

Puede comprobarse que el almacenamiento se produce correctamente ya que al realizar una nueva búsqueda el usuario no pierde la información ya clasificada sobre el referéndum del 15 de Octubre en Madrid.

De la misma manera que en la búsqueda anterior, el usuario clasifica en una nueva categoría la información que le interesa de los resultados obtenidos y le asigna el título 'Referéndum 15 de Octubre en Barcelona'.

Figura 3.25. Clasificación de los tweets en la categoría 'Referéndum 15 Octubre en Barcelona'.

El usuario ya tiene alguna información clasificada, pero al revisar la información de cada categoría se da cuenta de que en la categoría de Madrid hay información duplicada, por lo que decide eliminar los tweets que no le interesan arrastrándolos a la papelera, puesto que considera que la información de ese tweet no la va a necesitar más en esta clasificación.

Figura 3.26. Eliminación permanente de un tweet.

El usuario considera que ya tiene la información suficiente, y le gustaría tenerla en un fichero por lo que descarga el contenido de cada clasificación. Obtiene dos ficheros en formato 'pdf' en el que aparece el título asignado a categoría y los tweets clasificados en la misma.

Figura 3.27. Descarga del contenido clasificado.

Una vez ha clasificado la información, quiere saber más sobre un usuario que ha publicado un tweet con información de un video de la manifestación, así que decide meterse en su perfil para intentar ver dicho video.

Figura 3.28. Acceso al perfil Twitter de un usuario.

Por último, elimina todo el contenido de las clasificaciones para que la próxima vez que entre en la aplicación no encuentre ningún contenido ya descargado y poder empezar una nueva clasificación desde el principio.

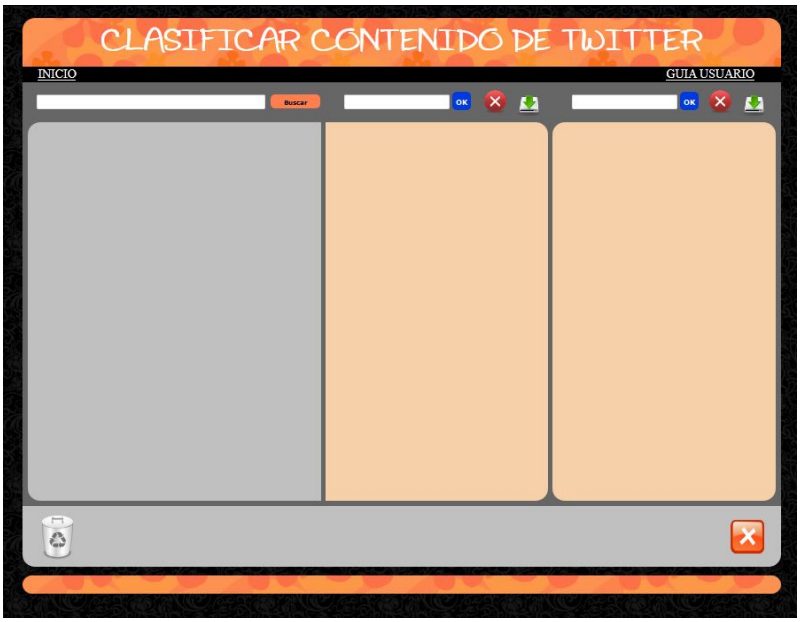


Figura 3.29. Eliminación de todo el contenido clasificado.

Conclusión

Esta descripción de verificación se trata de una validación sencilla del sistema pero no por ello menos eficaz, con ella se puede comprobar que la funcionalidad prevista de la aplicación se la implementado correctamente y cumple con los objetivos propuestos en el proyecto. Por lo que se puede concluir que el desarrollo de la aplicación ha sido exitoso.

Capítulo 4

Presupuesto

4. Presupuesto

El producto obtenido tras la realización de este proyecto consiste en una aplicación Web que permite la búsqueda, clasificación y almacenamiento de información de la red social Twitter. Esta implementada en HTML 5 y desarrollada bajo la plataforma Windows 7.

La aplicación se ha desarrollado para facilitar al usuario la filtración de la información así como dar la posibilidad de clasificarla y descargarla.

En este presupuesto se detallan los recursos tanto materiales como humanos que han sido necesarios a lo largo de los 5 meses de desarrollo de la aplicación, estos son: la persona que ha desarrollado la aplicación, el analista del proyecto, el ordenador de desarrollo y el equipo de análisis, y además se debe incluir el coste de contratación de un servidor Web durante un año, es este caso 'Apache' para asociar el dominio a la aplicación y poder acceder a ella a través de Internet.

Este presupuesto se podría ampliar para que contemple un proyecto de mayor alcance y complejidad, además de incluir un equipo de desarrollo mayor que agilizaría el tiempo de desarrollo.

4.1 Presupuesto general

El desarrollo de este proyecto se estima que durará 5 meses y supondrá un coste de 18.200 euros aproximadamente, en el que se han tenido en cuenta las amortizaciones de los equipos y materiales utilizados. A continuación se muestra un resumen del mismo:

Concepto	Descripción	Coste (€)
Recursos humanos	Estos recursos incluyen al programador/diseñador Web y al analista del proyecto.	15.000,00
Recursos materiales	Se refiere a los equipos y componentes asociados.	77,14
Otros	El servidor Web contratado	90,00
	Total (+ 20 % costes indirectos)	**18.200,57**

Tabla 4.1. Presupuesto general.

4.2 Presupuesto detallado

Autor: Marta Fuentes Lara

Departamento: Informática

Descripción del proyecto

Titulo → Aplicación Web para la selección y clasificación de contenido en la red social Twitter.

Duración (meses) → 5

Tasa de costes Indirectos → 20%

Desglose presupuestario (costes directos):

PERSONAL				
Apellidos y nombre	Categoría	Dedicación (meses)	Coste (hombre/mes[19])	Coste (€)
Fuentes Lara, Marta	Programador/ Diseñador Web Junior	5	1.750,00	8.750,00
Acuña Ruano, Pablo	Analista programador Senior	2,5	2.500,00	6.250,00
			Total	15.000,00

Tabla 4.2. Costes directos en personal.

EQUIPOS Y MATERIALES				
Descripción	Coste (€)	Dedicación (meses)	Vida útil (meses)	Coste imputable (€)
Ordenador Portátil de desarrollo	850,00	5,00	72	59,03
Ordenador de análisis	600,00	2,50	96	15,63
Ratón inalámbrico Microsoft	17,90	5,00	36	2,49
			Total	77,14

Tabla 4.3. Costes directos en equipo y material.

OTROS COSTES DIRECTOS DEL PROYECTO		
Descripción	Empresa	Coste imputable (€)
Servidor Web Apache	Apache	90,00
	Total	**90,00**

Tabla 4.4. Otros costes directos.

[19] Dato obtenido de: http://www.tufuncion.com/trabajo-programador

Resumen de costes

Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	15.000,00
Amortización	77,14
Subcontratación de tareas	0,00
Costes de funcionamiento	90,00
Costes Indirectos	3.033,43
Total	**18200,57**

Tabla 4.5. Resumen de costes.

Capítulo 5

Conclusiones

5. Conclusiones

5.1 Conclusiones principales

Al principio de este documento se fijaron los objetivos que se debían cumplir en la aplicación. Se recuerdan brevemente:

- Permitir la selección y clasificación de información de la red social Twitter.
- Permitir el almacenamiento temporal de la información clasificada sin necesidad de recargar la página.
- Recuperar la información almacenada automáticamente cuando se recargue la página.
- Fijar dos zonas de clasificación de información.
- Eliminar la información de una clasificación, de las dos o un tweet de manera individual.
- Permitir el almacenamiento en un fichero 'pdf'.

Toda la información que ha hecho posible llevar a cabo con éxito los objetivos se recoge en el Estado del arte (Capítulo 2). En este capítulo se analizan las diferentes API's que ofrece Twitter al desarrollador para permitir el acceso a la información de su base de datos principal y se llega a la conclusión de que 'Search API' se ajusta mucho mejor a las necesidades del proyecto, permitiendo realizar consultar sencillas pero eficaces.

Para cumplir con el segundo objetivo, también se analizaron las distintas posibilidades que ofrecían las tecnologías actuales y se descubrió que la nueva propiedad 'LocalStorage' de JavaScript se acoplaba perfectamente a HTML 5 y hace posible el almacenamiento de la información en el navegador en el que se ejecuta la aplicación. Por otro lado, para el almacenamiento de la información en un fichero se ha utilizado PHP y concretamente la librería 'fpdf[20]' con la que podemos crear y personalizar un fichero con formato '.pdf'.

La propiedad 'LocalStorage' también contribuye a cumplir el tercer objetivo ya que dispone de función 'getItem()' que se encarga de recuperar datos almacenados.

Con HTML 5 se pueden crear todo tipo de estructuras, por lo que con él se han creado las dos zonas que define las zonas de clasificación y dentro de las que se ha utilizado la técnica 'Drag and Drop' para crear listas que acepten la información a clasificar.

[20] http://www.fpdf.org/

El objetivo que consiste en permitir al usuario descartar la información que no le sea de utilidad se lleva a cabo a través de funciones JavaScript que se llaman dentro de la estructura HTML 5 en forma de botones. Aunque en el caso de la eliminación de un tweet individual se realiza de nuevo con la técnica 'Drag and Drop' con la que se logra arrastrar un tweet a la papelera para eliminarlo.

Por último, para ofrecer una ayuda al usuario se crea una Guía que contiene información específica sobre lo que puede hacer usando la aplicación y cómo puede hacerlo.

Para concluir, decir que aunque se han cumplido todos los objetivos fijados en algunos de ellos se han encontrado inconvenientes que han dificultado su implementación y que han tenido que solucionarse realizando modificaciones en el código:

- Una dificultad se presentó fue al momento de implementar el almacenamiento de la información en el navegador, ya que la información a almacenar era un array que contenía a cada tweet y cada tweet era otro array compuesto de los tres elementos definidos previamente (imagen, enlace y texto); sin embargo, 'LocalStorage' solo acepta almacenar cadenas. Por esta razón se tuvo que convertir el formato del array de arrays en una cadena, almacenar esa cadena, y cuando se recupera la información volver a convertir la cadena en el array original para mostrarla.
- Otro inconveniente surgió al crear el fichero '.pdf'. El problema surgió con las imágenes que, al obtenerse del API de Twitter, presentan formatos diferentes y la librería 'fpdf' utiliza un método distinto para escribir cada tipo de imagen, por lo que se tuvieron que convertir todas las imágenes al mismo formato ('jpeg') y después ir escribiéndolas en el fichero.

5.2 Conclusiones personales

Llevar a cabo este proyecto ha sido una de las mejores experiencias que he vivido a lo largo de la carrera. En él he podido aplicar mis conocimientos y aprender otros nuevos relacionados que me serán de utilidad en el futuro.

Además, debido a la cierta flexibilidad que ofrece la realización de un proyecto como éste, me he servido de mi creatividad para dar forma al diseño de la aplicación propuesta, que aunque es mejorable se adapta a la forma y apariencia que se buscaba.

Por otro lado, realizar un proyecto de esta envergadura me ha permitido mejorar mis habilidades en la documentación e investigación y profundizar en los aspectos de diseño y evaluación.

Han sido muchas horas de trabajo y esfuerzo pero estoy muy satisfecha con el resultado obtenido y el haber podido contribuir, aunque sea en una pequeña parte, ha mejorar el mundo de la informática me llena de orgullo y alegría.

5.3 Mejoras o trabajos futuros

A continuación se proponen una serie de mejoras que podrían realizarse en el prototipo para ofrecer una mayor funcionalidad y hacer más eficaz la aplicación. Estas acciones no se han implementado en el prototipo actual debido a limitaciones de tiempo y por no formar parte de la funcionalidad esencial buscada:

- **Ampliación de zonas de categorías:** se podrían añadir estructuras dinámicas que permitieran al usuario crear tantas zonas de categorías como necesitara, con solo pulsar un botón.
- **Permitir otros formatos en los ficheros:** podría ser de utilidad ofrecer al usuario la posibilidad de elegir el formato que tendrá el fichero que se descargará con la información clasificada. Algunos de estos formatos podrían ser: '.xml', '.txt', '.doc' o '.rtf'.
- **Clasificación colaborativa:** añadiendo una mayor complejidad al prototipo se podría permitir una clasificación colaborativa, es decir, que se permita clasificar una información entre varios usuarios desde distintos equipos. Esto supondría la implementación de métodos de sincronización que mantuvieran la integridad de los datos procesados.
- **Personalización de 'tweets' clasificados:** otra mejora podría ser que se permitiera personalizar la apariencia de los tweets clasificados en zonas tanto en la aplicación como en el documento generado, para que el usuario pueda cambiar la forma en la que se muestra la información si así lo desea. Esto último pudiera lograrse al generar un documento en un formato estándar como XML o JSON, permitiendo a aplicaciones terceras procesarlos y generar visualizaciones personalizadas.
- **Generación de un documento de descarga ampliado:** Podría ser del interés del usuario tener una opción con la que se generara un solo documento que contuviera la información de todas las clasificaciones, con el objetivo de tener toda la información junta.
- **Video-guía de usuario:** debido a la facilidad en la que en HTML 5 se pueden introducir videos podría ser más cómodo para el usuario que la guía fuera un video demostrativo.
- **Impresión directa:** aunque en todos los navegadores se puede descargar la información de la página que se está viendo, sería de utilidad que se ofreciera

un botón que permitiera imprimir directamente la información clasificada en las diferentes zonas de clasificación, sin necesidad de generar el fichero. Para esto se podría utilizar CSS, modificando el estilo y presentación del contenido clasificado permitiendo un formato más adecuado a un documento impreso.

BIBLIOGRAFÍA Y REFERENCIAS

- Bruce Lawson, Remy Sharp: 'Introducing HTML 5' (Berkeley, CA: New Riders, 2010)
- López Quijado, José: 'Domine JavaScript' (Paracuellos del Jarama (Madrid): Ra-Ma, D.L. 2007, 2ª edición, 2007)
- Negrino Tom, Smith, Dori: 'JavaScript and Ajax para diseño Web: guía de aprendizaje' (Pearson/Prentice Hall, cop. 2007, 6ª edición, 2007)
- López Quijado, José: 'Domine PHP 5' (Madrid: RA-MA, 2008)
- http://blog.mozilla.com/webdev/2009/02/12/native-json-in-firefox-31/
- http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/
- http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/
- http://html5doctor.com/introducing-web-sql-databases/
- http://paperkilledrock.com/2010/05/html5-localstorage-part-one/
- http://websitetips.com/dhtml/
- http://www.danilat.com/weblog/2010/05/18/sql-con-javascript-web-sql-database/
- http://www.ibm.com/developerworks/opensource/library/x-twitsrchapi/index.html
- http://www.json.org/js.html
- http://www.w3.org/standards/webdesign/htmlcss
- http://www.w3schools.com/css/css_pseudo_classes.asp
- http://www.webreference.com/programming/javascript/mk/column2/
- https://dev.twitter.com/docs/streaming-api/methods
- [1] Blorgia: <http://www.blorgia.com/articulos/actualidad/ que-es-html5-novedades-del-nuevo-standard-i/> [Septiembre 2011]
- [2] Webistar: <http://webistar.es/inicio/item/38-contenteditable-contenido-editable-en-html5> [Septiembre 2011]
- [3] Tutoriales HTML 5: <http://www.tutorialeshtml5.com/2010/06/formulario-en-html-5-i.html> [Septiembre 2011]
- [4] Desarrollo web: <http://www.desarrolloweb.com/manuales/css3.html> [Septiembre 2011]

- [5] Taringa: <http://www.taringa.net/posts/info/11302933/Curso-completo-de-CSS3.html> [Septiembre 2011]
- [6] Desarrollo Web: <http://www.desarrolloweb.com/articulos/sombras-css3-box-shadow.html> [Septiembre 2011]
- [7] Applications using Webkit: <http://trac.webkit.org/wiki/Applications%20using%20WebKit>[Septiembre 2011]
- [8] Libros web: <http://www.librosweb.es/css_avanzado/capitulo3/ selectores_de_css_3.html> [Septiembre 2011]
- [9] Libros web: <http://www.librosweb.es/css_avanzado/capitulo3/ pseudo-clases.html> [Septiembre 2011]
- [10] Igniside.net: <http://www.ignside.net/man/css/clases.php> [Septiembre 2011]
- [11] Libros web: <http://www.librosweb.es/ajax/capitulo4.html> [Septiembre 2011]
- [12] Libros web: <http://www.librosweb.es/ajax/capitulo4/html_y_dom.html> [Septiembre 2011]
- [13] Desarrollo web: <http://www.desarrolloweb.com/articulos/449.php> [Septiembre 2011]
- [14] Tutorial sobre que es JSON- Para tu pagina: <http://paratupagina.com/topic/395-tutorial-sobre-que-es-json/> [Septiembre 2011]
- [15] Using native JSON-MDN: <https://developer.mozilla.org/En/Using_native_JSON> [Septiembre 2011]
- [16] HTML 5-Almacenamiento de datos en el cliente| Pixelco: <http://pixelcoblog.com/html5-almacenamiento-de-datos-en-el-cliente/> [Septiembre 2011]
- [17] Introducing Web SQL Databases-HTML 5 Doctor: <http://html5doctor.com/introducing-web-sql-databases/> [Septiembre 2011]
- [18] MDN: Using IndexedDB: <https://developer.mozilla.org/en/IndexedDB/IndexedDB_primer> [Septiembre 2011]
- [19] Blog de Formato web.com.ar: <http://www.formatoweb.com.ar/blog/2007/09/22/drag-drop-arrastrar-y-soltar-simple-en-un-div-con-javascript-sin-librerias> [Septiembre 2011]

- [20] Borasky Research Journal: <http://borasky-research.net/2010/01/06/the-twitter-streaming-api-how-it-works-and-why-its-a-big-deal/> [Septiembre 2011]
- [21] Using the Twitter REST API- IBM: <http://www.ibm.com/developerworks/xml/library/x-twitterREST/> [Septiembre 2011]
- [22] Build a RESTful Web service- IBM: <http://www.ibm.com/developerworks/java/tutorials/j-rest/section2.html> [Septiembre 2011]
- [23] Streaming API | Twitter Developers: <http://dev.twitter.com/pages/streaming_api> [Septiembre 2011]
- [24] Streaming API Methods | Twitter Developers: <https://dev.twitter.com/docs/streaming-api/methods> [Septiembre 2011]
- [25] PHP Twitter Search API: <http://ryanfaerman.com/twittersearch/> [Septiembre 2011]
- [26] Atom (standard) - Wikipedia, the free encyclopedia: <http://en.wikipedia.org/wiki/Atom_(standard)> [Septiembre 2011]
- [27] JSON: <http://www.json.org/> [Septiembre 2011]
- [28] JSONP: JSON con Padding: <http://blog.ikhuerta.com/jsonp-json-con-padding> [Septiembre 2011]
- [29] Apache friends – xampp: <http://www.apachefriends.org/en/xampp.html> [Octubre 2011]
- [30] Los 10 países más adictos a Twitter en el mundo-ALT1040 : <http://alt1040.com/2011/04/los-10-paises-mas-adictos-a-twitter> [Septiembre 2011]

Anexo A: Siglas y Acrónimos

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ATOM	Any Transport over MPLS
CSS	Cascading Style Sheet
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IndexedDB	Indexed data base
JS	JavaScript
JSON	JavaScript Object Notation
MPLS	Multi-Protocol Label Switching
Native JSON	Native JavaScript Object Notation
PDF	Portable document format
PHP	Hypertext Pre-processor
REST	Representational State Transfer
SQL	Structured Query Language
TDT	Técnica de Detección y Seguimiento de Tema
URL	Uniform Resource Locator
XHTML	eXtensible Hypertext Markup Language
XML	Extensive Markup Language